1. Implement any two Substitution Techniques. AIM: Write a Java program to perform encryption and decryption using the  
   following algorithms:  
   a) **Ceaser Cipher**

ALGORITHM

1. In Ceaser Cipher each letter in the plaintext is replaced by a letter some

fixed number of positions down the alphabet.

2. For example, with a left shift of 3, D would be replaced by A, E would

become B, and so on.

3. The encryption can also be represented using modular arithmetic by first

transforming the letters into numbers, according to the scheme, A = 0, B = 1,Z = 25.

4. Encryption of a letter x by a shift n can be described mathematically as,

En(x) = (x + n) mod26

5. Decryption is performed similarly,

Dn (x)=(x - n) mod26

PROGRAM:

class caesarCipher {

public static String encode(String enc, int offset) {

offset = offset % 26 + 26;

StringBuilder encoded = new StringBuilder();

for (char i : enc.toCharArray()) {

if (Character.isLetter(i)) {

if (Character.isUpperCase(i)) {

encoded.append((char) ('A' + (i - 'A' + offset) % 26));

} else {

encoded.append((char) ('a' + (i - 'a' + offset) % 26));

}

} else {

encoded.append(i);

}

}

return encoded.toString();

}

public static String decode(String enc, int offset) {

return encode(enc, 26 - offset);

}

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

String msg = "LBRCE CSE";

System.out.println("Simulating Caesar Cipher\n------------------------");

System.out.println("Input : " + msg);

System.out.printf("Encrypted Message : ");

System.out.println(caesarCipher.encode(msg, 3));

System.out.printf("Decrypted Message : ");

System.out.println(caesarCipher.decode(caesarCipher.encode(msg, 3), 3));

}

}

Output:

Simulating Caesar Cipher

------------------------Input : LBRCE CSE

Encrypted Message : OEUFH FVH

Decrypted Message : LBRCE CSE

**RESULT:**

Thus the program for ceaser cipher encryption and decryption algorithm has

been implemented and the output verified successfully.

b) **Playfair Cipher**

**AIM:**

To implement a program to encrypt a plain text and decrypt a cipher text

using play fair Cipher substitution technique.

**ALGORITHM:**

1. To encrypt a message, one would break the message into digrams (groups of

2 letters)

2. For example, "HelloWorld" becomes "HE LL OW OR LD".

3. These digrams will be substituted using the key table.

4. Since encryption requires pairs of letters, messages with an odd number of

characters usually append an uncommon letter, such as "X", to complete the

final digram.

5. The two letters of the digram are considered opposite corners of a rectangle

in the key table. To perform the substitution, apply the following 4 rules, in

order, to each pair of letters in the plaintext:

**PROGRAM:**

**playfairCipher.java**

import java.awt.Point;

class playfairCipher {

private static char[][] charTable;

private static Point[] positions;

private static String prepareText(String s, boolean chgJtoI) {

s = s.toUpperCase().replaceAll("[^A-Z]", "");

return chgJtoI ? s.replace("J", "I") : s.replace("Q", "");

}

private static void createTbl(String key, boolean chgJtoI) {

charTable = new char[5][5];

positions = new Point[26];

String s = prepareText(key + "ABCDEFGHIJKLMNOPQRSTUVWXYZ",

chgJtoI);

int len = s.length();

for (int i = 0, k = 0; i < len; i++) {

char c = s.charAt(i);

if (positions[c - 'A'] == null) {

charTable[k / 5][k % 5] = c;

positions[c - 'A'] = new Point(k % 5, k / 5);

k++;

}

}

}

private static String codec(StringBuilder txt, int dir) {

int len = txt.length();

for (int i = 0; i < len; i += 2) {

char a = txt.charAt(i);

char b = txt.charAt(i + 1);

int row1 = positions[a - 'A'].y;

int row2 = positions[b - 'A'].y;

int col1 = positions[a - 'A'].x;

int col2 = positions[b - 'A'].x;

if (row1 == row2) {

col1 = (col1 + dir) % 5;

col2 = (col2 + dir) % 5;

} else if (col1 == col2) {

row1 = (row1 + dir) % 5;

row2 = (row2 + dir) % 5;

} else {

int tmp = col1;

col1 = col2;

col2 = tmp;

}

txt.setCharAt(i, charTable[row1][col1]);

txt.setCharAt(i + 1, charTable[row2][col2]);

}

return txt.toString();

}

private static String encode(String s) {

StringBuilder sb = new StringBuilder(s);

for (int i = 0; i < sb.length(); i += 2) {

if (i == sb.length() - 1) {

sb.append(sb.length() % 2 == 1 ? 'X' : "");

} else if (sb.charAt(i) == sb.charAt(i + 1)) {

sb.insert(i + 1, 'X');

}

}

return codec(sb, 1);

}

private static String decode(String s) {

return codec(new StringBuilder(s), 4);

}

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

String key = "CSE";

String txt = "Security Lab"; /\* make sure string length is even \*/ /\* change J

to I \*/

boolean chgJtoI = true;

createTbl(key, chgJtoI);

String enc = encode(prepareText(txt, chgJtoI));

System.out.println("Simulating Playfair Cipher\n----------------------");

System.out.println("Input Message : " + txt);

System.out.println("Encrypted Message : " + enc);

System.out.println("Decrypted Message : " + decode(enc));

}

}

**OUTPUT:**

Simulating Playfair Cipher

----------------------

Input Message : Security Lab

Encrypted Message : EABPUGYANSEZ

Decrypted Message : SECURITYLABX

**RESULT:**

Thus the program for playfair cipher encryption and decryption algorithm

has been implemented and the output verified successfully.

2. Implement any two Transposition Techniques

a) Rail Fence Cipher Transposition Technique

**AIM:**

To implement a program for encryption and decryption using rail fence

transposition technique.

**ALGORITHM:**

1. In the rail fence cipher, the plaintext is written downwards and diagonally on

successive "rails" of an imaginary fence, then moving up when we reach the

bottom rail.

2. When we reach the top rail, the message is written downwards again until

the whole plaintext is written out.

3. The message is then read off in rows.

**PROGRAM:**

**railFenceCipher.java**

class railfenceCipherHelper {

int depth;

String encode(String msg, int depth) throws Exception {

int r = depth;

int l = msg.length();

int c = l / depth;

int k = 0;

char mat[][] = new char[r][c];

String enc = "";

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

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

if (k != l) {

mat[j][i] = msg.charAt(k++);

} else {

mat[j][i] = 'X';

}

}

}

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

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

enc += mat[i][j];

}

}

return enc;

}

String decode(String encmsg, int depth) throws Exception {

int r = depth;

int l = encmsg.length();

int c = l / depth;

int k = 0;

char mat[][] = new char[r][c];

String dec = "";

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

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

mat[i][j] = encmsg.charAt(k++);

}

}

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

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

dec += mat[j][i];

}

}

return dec;

}

}

class railFenceCipher {

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

railfenceCipherHelper rf = new railfenceCipherHelper();

String msg, enc, dec;

msg = "INFORMATION SECURITY";

int depth = 2;

enc = rf.encode(msg, depth);

dec = rf.decode(enc, depth);

System.out.println("Simulating Railfence Cipher\n-------------------------");

System.out.println("Input Message : " + msg);

System.out.println("Encrypted Message : " + enc);

System.out.printf("Decrypted Message : " + dec);

}

}

**OUTPUT:**

Simulating Railfence Cipher

-------------------------

Input Message : INFORMATION SECURITY

Encrypted Message : IFRAINSCRTNOMTO EUIY

Decrypted Message : INFORMATION SECURITY

**RESULT:**

Thus the java program for Rail Fence Transposition Technique has been

implemented and the output verified successfully.

**b) Columnar Transformation Technique**

**AIM:**

To implement a program for encryption and decryption by using row and

column transformation technique.

**ALGORITHM:**

1. Consider the plain text hello world, and let us apply the simple columnar

transposition technique as shown below

|  |  |  |  |
| --- | --- | --- | --- |
| H | e | l | l |
| O | w | o | r |
| L | d |  |  |

2. The plain text characters are placed horizontally and the cipher text is

created with vertical format as: holewdlo lr.

3. Now, the receiver has to use the same table to decrypt the cipher text to

plain text.

**PROGRAM:**

**TransCipher.java**

import java.util.\*;

class TransCipher {

public static void main(String args[]) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the plain text");

String pl = sc.nextLine();

sc.close();

String s = "";

int start = 0;

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

if (pl.charAt(i) == ' ') {

s = s + pl.substring(start, i);

start = i + 1;

}

}

s = s + pl.substring(start);

System.out.print(s);

System.out.println();

// end of space deletion

int k = s.length();

int l = 0;

int col = 4;

int row = s.length() / col;

char ch[][] = new char[row][col];

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

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

if (l < k) {

ch[i][j] = s.charAt(l);

l++;

} else {

ch[i][j] = '#';

}

}

}

// arranged in matrix

char trans[][] = new char[col][row];

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

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

trans[j][i] = ch[i][j];

}

}

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

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

System.out.print(trans[i][j]);

}

}

// display

System.out.println();

}

}

**OUTPUT:**

Enter the plain text

information security

informationsecurity

irienmocfanuotsr

**RESULT:**

Thus the java program for Row and Column Transposition Technique has been implemented and the output verified successfully.

1. Implement any two Symmetric algorithms.

Aim: To implement the Advanced encryption standard(Aes)

Algorithm:

The AES algorithm uses a substitution-permutation, or SP network, with multiple rounds to produce ciphertext. The number of rounds depends on the key size being used. A 128-bit key size dictates ten rounds, a 192-bit key size dictates 12 rounds, and a 256-bit key size has 14 rounds. Each of these rounds requires a round key, but since only one key is inputted into the algorithm, this key needs to be expanded to get keys for each round, including round 0.

Each round in the algorithm consists of four steps.

1. Substitution of the bytes

In the first step, the bytes of the block text are substituted based on rules dictated by predefined S-boxes (short for substitution boxes)

2. Shifting the rows

Next comes the permutation step. In this step, all rows except the first are shifted by one, as shown below.

3. Mixing the columns

In the third step, the Hill cipher is used to jumble up the message more by mixing the block’s columns.

4. Adding the round key

In the final step, the message is XORed with the respective round key.

When done repeatedly, these steps ensure that the final ciphertext is secure.

**PROGRAM:**

**Aes.java**

import java.io.UnsupportedEncodingException;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.util.Arrays;

import java.util.Base64;

import javax.crypto.Cipher;

import javax.crypto.spec.SecretKeySpec;

public class AES {

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);

secretKey = new SecretKeySpec(key, "AES");

} catch (NoSuchAlgorithmException e) {

e.printStackTrace();

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

}

}

public static String encrypt(String strToEncrypt, String secret) {

try {

setKey(secret);

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

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

return

Base64.getEncoder().encodeToString(cipher.doFinal(strToEncrypt.getBytes("UTF-8")));

} catch (Exception e) {

System.out.println("Error while encrypting: " + e.toString());

}

return null;

}

public static String decrypt(String strToDecrypt, String secret) {

try {

setKey(secret);

Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");

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

return new

String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));

} catch (Exception e) {

System.out.println("Error while decrypting: " + e.toString());

}

return null;

}

public static void main(String[] args) {

final String secretKey = "lbrce csection";

String originalString = "www.lbrce.edu";

String encryptedString = AES.encrypt(originalString, secretKey);

String decryptedString = AES.decrypt(encryptedString, secretKey);

System.out.println("URL Encryption Using AES Algorithm\n------------");

System.out.println("Original URL : " + originalString);

System.out.println("Encrypted URL : " + encryptedString);

System.out.println("Decrypted URL : " + decryptedString);

}

}

**OUTPUT:**

URL Encryption Using AES Algorithm

------------

Original URL : www.lbrce.edu

Encrypted URL : 77eTRluGI5G3/VxvVlhc7A==

Decrypted URL : [www.lbrce.edu](http://www.lbrce.edu)

**RESULT:**

Thus the java program for AES has been implemented and the output verified successfully.

**b)DES(Data Encryption Standard)**

## Aim: Simple D.E.S Implementation

### Description:

Simplified DES, developed by Professor Edward Schaefer of Santa Clara University is an educational rather than a secure encryption algorithm. It has similar properties and structure to DES with much smaller parameters.

|  |  |  |  |
| --- | --- | --- | --- |
| S-DES encryption algorithm | | S-DES decryption algorithm | |
| Input | output | input | output |
| * 8-bit block of plaintext * 10-bit key | * 8-bit block of ciphertext | * 8-bit block of ciphertext * 10-bit key | * original 8-bit block of plaintext |

The encryption algorithm involves five functions:

* an initial permutation (IP);
* a complex function labeled fK, which involves both permutation and substitution operations and depends on a key input;
* a simple permutation function that switches (SW) the two halves of the data;
* the function fK again; and
* finally a permutation function that is the inverse of the initial permutation (IP–1).



which can also be written as:



Where

*K*1 P8ShiftP10key

*K*2 P8ShiftShiftP10key

Decryption is also shown in Figure G.1 and is essentially the reverse of encryption:



**Code:**

import java.util.\*;

import javax.crypto.BadPaddingException;

import javax.crypto.Cipher;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.KeyGenerator;

import javax.crypto.NoSuchPaddingException;

import javax.crypto.SecretKey;

import javax.crypto.SecretKeyFactory;

import javax.crypto.spec.DESKeySpec;

import java.io.\*;

import java.security.InvalidKeyException;

import java.security.NoSuchAlgorithmException;

import java.security.spec.InvalidKeySpecException;

class DES{

    public static void main(String[] args) throws IOException, NoSuchAlgorithmException, InvalidKeyException, InvalidKeySpecException, NoSuchPaddingException, IllegalBlockSizeException, BadPaddingException {

        //String we want to encrypt

        String message="This is a confidential message.";

        byte[] myMessage =message.getBytes(); //string to byte array as DES works on bytes

        //If you want to use your own key

        // SecretKeyFactory MyKeyFactory = SecretKeyFactory.getInstance("DES");

        // String Password = "My Password";

        // byte[] mybyte =Password.getBytes();

        // DESKeySpec myMaterial = new DESKeySpec(mybyte);

        // SecretKey myDESKey = MyKeyFactory.generateSecret(myMaterial);

        //Generating Key

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

        SecretKey myDesKey = Mygenerator.generateKey();

        //initializing crypto algorithm

        Cipher myCipher = Cipher.getInstance("DES");

        //setting encryption mode

        myCipher.init(Cipher.ENCRYPT\_MODE, myDesKey);

        byte[] myEncryptedBytes=myCipher.doFinal(myMessage);

        //setting decryption mode

        myCipher.init(Cipher.DECRYPT\_MODE, myDesKey);

        byte[] myDecryptedBytes=myCipher.doFinal(myEncryptedBytes);

        //print message in byte format

        //System.out.println(Arrays.toString(myEncryptedBytes));

        //System.out.println(Arrays.toString(myDecryptedBytes));

        String encrypteddata=new String(myEncryptedBytes);

        String decrypteddata=new String(myDecryptedBytes);

        System.out.println("Message : "+ message);

        System.out.println("Encrypted - "+ encrypteddata);

        System.out.println("Decrypted Message - "+ decrypteddata);

    }

}

4. Implement any two Public-Key based algorithms

a) Diffie-Hellman Key Exchange algorithm

AIM:

To implement the Diffie-Hellman Key Exchange algorithm for a given

problem .

ALGORITHM:

1. Alice and Bob publicly agree to use a modulus p = 23 and base g = 5 (which

is a primitive root modulo 23).

2. Alice chooses a secret integer a = 4, then sends Bob A = ga mod p

* A = 54 mod 23 = 4

3. Bob chooses a secret integer b = 3, then sends Alice B = gb mod p

* B = 53 mod 23 = 10

4. Alice computes s = Ba mod p

* s = 104 mod 23 = 18

5. Bob computes s = Ab mod p

* s = 43 mod 23 = 18

6. Alice and Bob now share a secret (the number 18).

PROGRAM:

DiffieHellman.java

class DiffieHellman {

public static void main(String args[]) {

int p = 23; /\* publicly known (prime number) \*/

int g = 5; /\* publicly known (primitive root) \*/

int x = 4; /\* only Alice knows this secret \*/

int y = 3; /\* only Bob knows this secret \*/

double aliceSends = (Math.pow(g, x)) % p;

double bobComputes = (Math.pow(aliceSends, y)) % p;

double bobSends = (Math.pow(g, y)) % p;

double aliceComputes = (Math.pow(bobSends, x)) % p;

double sharedSecret = (Math.pow(g, (x \* y))) % p;

System.out.println("simulation of Diffie-Hellman key exchange algorithm\n---------------------------------------------");

System.out.println("Alice Sends : " + aliceSends);

System.out.println("Bob Computes : " + bobComputes);

System.out.println("Bob Sends : " + bobSends);

System.out.println("Alice Computes : " + aliceComputes);

System.out.println("Shared Secret : " + sharedSecret);

/\* shared secrets should match and equality is transitive \*/

if ((aliceComputes == sharedSecret) && (aliceComputes == bobComputes))

System.out.println("Success: Shared Secrets Matches! " + sharedSecret);

else

System.out.println("Error: Shared Secrets does not Match");

}

}

OUTPUT:

simulation of Diffie-Hellman key exchange algorithm

-----------------------------------------------------------------

Alice Sends : 4.0

Bob Computes : 18.0

Bob Sends : 10.0

Alice Computes : 18.0

Shared Secret : 18.0

Success: Shared Secrets Matches! 18.0

RESULT:

Thus the Diffie-Hellman key exchange algorithm has been implemented

using Java Program and the output has been verified successfully.

b) RSA

AIM:

To Calculate the message digest of a text using the RSA algorithm.

ALGORITHM:



**Code:**

import java.math.\*;

import java.util.\*;

class RSA {

    public static void main(String args[])

    {

        int p, q, n, z, d = 0, e, i;

        // The number to be encrypted and decrypted

        int msg = 12;

        double c;

        BigInteger msgback;

        // 1st prime number p

        p = 3;

        // 2nd prime number q

        q = 11;

        n = p \* q;

        z = (p - 1) \* (q - 1);

        System.out.println("the value of z = " + z);

        for (e = 2; e < z; e++) {

            // e is for public key exponent

            if (gcd(e, z) == 1) {

                break;

            }

        }

        System.out.println("the value of e = " + e);

        for (i = 0; i <= 9; i++) {

            int x = 1 + (i \* z);

            // d is for private key exponent

            if (x % e == 0) {

                d = x / e;

                break;

            }

        }

        System.out.println("the value of d = " + d);

        c = (Math.pow(msg, e)) % n;

        System.out.println("Encrypted message is : " + c);

        // converting int value of n to BigInteger

        BigInteger N = BigInteger.valueOf(n);

        // converting float value of c to BigInteger

        BigInteger C = BigDecimal.valueOf(c).toBigInteger();

        msgback = (C.pow(d)).mod(N);

        System.out.println("Decrypted message is : "

                           + msgback);

    }

    static int gcd(int e, int z)

    {

        if (e == 0)

            return z;

        else

            return gcd(z % e, e);

    }

}