**CAESAR CIPHER:**

**ALGORITHM:**

STEP 1: Caesar cipher is an example of a substitution cipher in which plaintext letters in the original message are replaced (substituted for) by cipher text letters

STEP 2: The easiest way to understand this is to consider that there are two alphabets:

PLAIN\_ALPHABET: ABCDEFGHIJKLMNOPQRSTUVWXYZ

CIPHER\_ALPHABET: DEFGHIJKLMNOPQRSTUVWXYZABC

STEP 3: The cipher alphabet is a shifted version of the plain alphabet. In this case, each letter in the cipher alphabet has to be shifted by 3 places to the right

STEP 4: The shift -- ( i.e., the number 3 ) is the secret key which must be shared by Alice and Bob if they want to send secret messages using this cipher

STEP 5: To encrypt the message MEET ME AT THE DOCK we would replace all the Ms in the message with the corresponding letter from the cipher alphabet

STEP 6: So M is replaced by P. And we would replace all the Es by H and so on. Thus, the encryption of our message would be PHHW PH DW WLH GRFN

**Algorithm:**

STEP 1. The playfair cipher was the first practical digraph substitution cipher. The technique encrypts

pairs of letters (digraphs), instead of single letters as in the simple substitution cipher

STEP 2. The 'key' for a playfair cipher is generally a word, for the sake of example we will choose

'monarchy'. This is then used to generate a 'key square', e.g.

m o n a r

c h y b d

e f g i k

l p q s t

u v w x z

STEP 3. Any sequence of 25 letters can be used as a key, so long as all letters are in it and there are no

repeats. Note that there is no 'j', it is combined with 'i'. We now apply the encryption rules to

encrypt the plaintext

i) Remove any punctuation or characters that are not present in the key square (this may mean

spelling out numbers, punctuation etc.)

ii) Identify any double letters in the plaintext and replace the second occurrence with an 'x' e.g.

'hammer' -> 'hamxer'

iii) If the plaintext has an odd number of characters, append an 'x' to the end to make it even

iv) Break the plaintext into pairs of letters, e.g. 'hamxer' -> 'ha mx er'

v) The algorithm now works on each of the letter pairs

vi) Locate the letters in the key square, (the examples given are using the key square above)

a. If the letters are in different rows and columns, replace the pair with the letters on

the same row respectively but at the other pair of corners of the rectangle defined by

the original pair. The order is important – the first encrypted letter of the pair is the

one that lies on the same row as the first plaintext letter. 'ha' -> 'bo', 'es' -> 'il'

b. If the letters appear on the same row of the table, replace them with the letters to

their immediate right respectively (wrapping around to the left side of the row if a

letter in the original pair was on the right side of the row). 'ma' -> 'or', 'lp' -> 'pq'

**c.** If the letters appear on the same column of the table, replace them with the letters

immediately below respectively (wrapping around to the top side of the column if a

letter in the original pair was on the bottom side of the column). 'rk' -> 'dt', 'pv' ->

'vo'

Hill Cipher Algorithm:

1. In a Hill cipher encryption, the plaintext message is broken up into blocks of length according to the matrix chosen

2. Each block of plaintext letters is then converted into a vector of numbers and is dotted with the

matrix

3. The results are then converted back to letters and the ciphertext message is produced

4. For decryption of the ciphertext message, the inverse of the encryption matrix must be found

Once found, the decryption matrix is then dotted with each -block of ciphertext, producing the

plaintext message.

c) Hill Cipher

**Program**

import javax.swing.JOptionPane;

public class HillCipher

{

//the 3x3 key matrix for 3 characters at once

public static int[][] keymat = newint[][]

{

{ 1, 2, 1 },

{ 2, 3, 2 },

{ 2, 2, 1 },

};

public static int[][] invkeymat = newint[][]{

{ -1, 0, 1 },

{ 2, -1, 0 },

{ -2, 2, -1},

};

public static String key = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

public static void main(String[] args)

{

// TODO code application logic here

String text,outtext ="";

int ch, n;

ch = Integer.parseInt(JOptionPane.showInputDialog(null, "Enter 1 to Encrypt and 2 to Decrypt!"));

text = JOptionPane.showInputDialog(null, "Enter plain/cipher text to encrypt?");

text = text.toUpperCase();

text = text.replaceAll("\\s",""); //removing spaces

n = text.length() % 3;

if(n!=0)

{

for(int i = 1; i<= (3-n);i++)

{

text+= 'X';

}

}

System.out.println("Padded Text:" + text);

char[] ptextchars = text.toCharArray();

switch(ch)

{

case 1:

for(int i=0;i<text.length(); i+=3)

{

outtext += encrypt(ptextchars[i],ptextchars[i+1],ptextchars[i+2]);

}

break;

case 2:

for(int i=0;i<text.length(); i+=3)

{

outtext += decrypt(ptextchars[i],ptextchars[i+1],ptextchars[i+2]);

}

break;

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

}

System.out.println("Output: " + outtext);

}

private static String encrypt(char a, char b, char c)

{

String ret = "";

int x,y, z;

int posa = (int)a - 65;

int posb = (int)b - 65;

int posc = (int)c - 65;

x = posa \* keymat[0][0] + posb \* keymat[1][0] + posc \* keymat[2][0];

y = posa \* keymat[0][1] + posb \* keymat[1][1] + posc \* keymat[2][1];

z = posa \* keymat[0][2] + posb \* keymat[1][2] + posc \* keymat[2][2];

a = key.charAt(x%26);

b = key.charAt(y%26);

c = key.charAt(z%26);

ret = "" + a + b + c;

return ret;

}

private static String decrypt(char a, char b, char c)

{

String ret = "";

int x,y,z;

int posa = (int)a - 65;

int posb = (int)b - 65;

int posc = (int)c - 65;

x = posa \* invkeymat[0][0]+ posb \* invkeymat[1][0] + posc \* invkeymat[2][0];

y = posa \* invkeymat[0][1]+ posb \* invkeymat[1][1] + posc \* invkeymat[2][1];

z = posa \* invkeymat[0][2]+ posb \* invkeymat[1][2] + posc \* invkeymat[2][2];

a = key.charAt((x%26<0)?(26+x%26):(x%26));

b = key.charAt((y%26<0)?(26+y%26):(y%26));

c = key.charAt((z%26<0)?(26+z%26):(z%26));

ret = "" + a + b + c;

return ret;

}

}

Vignere Cipher Algorithm:

1. A Vignere Square or Vignere table consists of the alphabet written out 26 times in

different rows, each alphabet shifted cyclically to the left compared to the previous alphabet,

corresponding to the 26 possible Caesar ciphers

2. At different points in the encryption process, the cipher uses a different alphabet from one

of the rows. The alphabet used at each point depends on a repeating keyword

3. The person sending the message to be encrypted (eg. attackatdawn) chooses a keyword

and repeats it until it matches the length of the plaintext, for example, the keyword lemon,

the cipher key will be lemonlemonle

4. Using a VignereSquare and a CipherKey each row starts with a key letter. The remainder

of the row holds the letters A to Z (in shifted order)

5. Although there are 26 key rows shown, you will only use as many keys (different

alphabets) as there are unique letters in the key string, here just 5 keys, {L, E, M, O, N}

6. For successive letters of the message, we are going to take successive letters of the key

string, and encipher each message letter using its corresponding key row. Choose the next

letter of the key, go along that row to find the column heading that matches the message

character; the letter at the intersection of [key-row, msg-col] is the enciphered letter

7. The first letter of the plaintext, A, is paired with L, the first letter of the key. So use row L

and column A of the Vignere square, namely L. Similarly, for the second letter of the

plaintext, the second letter of the key is used; the letter at row E and column T is X. The rest

of the plaintext is enciphered in a similar fashion

**Program**

public classVigenereCipher

{

public static String encrypt(String text, final String key)

{

String res = "";

text = text.toUpperCase();

for(int i = 0, j = 0; i<text.length(); i++)

{

char c = text.charAt(i);

if(c < 'A' || c > 'Z')

continue;

res += (char) ((c + key.charAt(j) - 2 \* 'A') % 26 + 'A');

j = ++j % key.length();

}

return res;

}

public static String decrypt(String text, final String key)

{

String res = "";

text = text.toUpperCase();

for(int i = 0, j = 0; i<text.length(); i++)

{

char c = text.charAt(i);

if(c < 'A' || c > 'Z')

continue;

res += (char) ((c - key.charAt(j) + 26) % 26 + 'A');

j = ++j % key.length();

}

return res;

}

public static void main(String[] args)

{

String key = "VIGENERECIPHER";

String message = "Welcome to PRIYADHARSHINI ENGINEERING COLLEGE";

String encryptedMsg = encrypt(message, key);

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

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

System.out.println("Decrypted message: " + decrypt(encryptedMsg, key));

}

}

Rail Fence – Row & Column Transformation

Aim: To write a Java program to implement rail fence algorithm 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. The message is then read off in rows
3. Write down the plain text message as a sequence of diagonals
4. Read the plain text written in Step 1 as a sequence of rows

Example: Original plain text message: Come home tomorrow

Arrange the plain text message as sequence of diagonals

**Program**

package columnar;

import java.io.\*;

public class Columnar

{

char arr[][],encrypt[][],decrypt[][],keya[],keytemp[];

public void creatematrixE(String s,Stringkey,int row,int column)

{

arr=newchar[row][column];

int k=0; keya=key.toCharArray();

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

{

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

{

if(k<s.length())

{

arr[i][j]=s.charAt(k); k++;

}

else

{

arr[i][j]=' ';

}

}

}

}

public void createkey(String key,int column)

{

keytemp=key.toCharArray();

for(int i=0;i<column-1;i++)

{

for(int j=i+1;j<column;j++)

{

if(keytemp[i]>keytemp[j])

{

char temp=keytemp[i]; keytemp[i]=keytemp[j]; keytemp[j]=temp;

}

}

}

}

public void creatematrixD(String s,Stringkey,int row,int column)

{

arr=newchar[row][column];

int k=0; keya=key.toCharArray();

for(int i=0;i<column;i++)

{

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

{

if(k<s.length())

{

arr[j][i]=s.charAt(k); k++;

}

else

{

arr[j][i]=' ';

}

}

}

}

public void encrypt(int row,int column)

{

encrypt=newchar[row][column];

for(int i=0;i<column;i++)

{

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

{

if(keya[i]==keytemp[j])

{

for(int k=0;k<row;k++)

{

encrypt[k][j]=arr[k][i];

}

keytemp[j]='?';

break;

}

}

}

}

public void decrypt(int row,int column)

{

decrypt=newchar[row][column];

for(int i=0;i<column;i++)

{

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

{

if(keya[j]==keytemp[i])

{

for(int k=0;k<row;k++)

{

decrypt[k][j]=arr[k][i];

}

keya[j]='?';

break;

}

}

}

}

public void resultE(int row,int column,char arr[][])

{

System.out.println("Result:");

for(int i=0;i<column;i++)

{

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

{

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

}

}

}

public void resultD(int row,int column,char arr[][])

{

System.out.println("Result:");

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

{

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

{

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

}

}

}

public static void main(String args[])throwsIOException

{

int row,column,choice;

Columnar obj=newColumnar();

BufferedReader in=newBufferedReader(newInputStreamReader(System.in));

System.out.println("Menu:\n1) Encryption\n2) Decryption");

choice=Integer.parseInt(in.readLine());

System.out.println("Enter the string:");

String s=in.readLine();

System.out.println("Enter the key:");

String key=in.readLine();

row=s.length()/key.length();

if(s.length()%key.length()!=0)

row++;

column=key.length();

switch(choice)

{

case 1:

obj.creatematrixE(s,key,row,column);

obj.createkey(key,column);

obj.encrypt(row,column);

obj.resultE(row,column,obj.encrypt);

break;

case 2:

obj.creatematrixD(s,key,row,column);

obj.createkey(key,column);

obj.decrypt(row,column);

obj.resultD(row,column,obj.decrypt);

break;

}

}

}

DES

Aim:

To write a Java program to implement DES algorithm

Algorithm:

1. Firstly, we need to process the key

2. Get a 64-bit key from the user. (Every 8th bit is considered a parity bit. For a key to have correct

parity, each byte should contain an odd number of "1" bits.)

3. Calculate the key schedule

4. Perform the following permutation on the 64-bit key

5. Split the permuted key into two halves. The first 28 bits are called C[0] and the last 28 bits are

called D[0]

6. Calculate the 16 subkeys. Start with i = 1. Perform one or two circular left shifts on both C[i-1]

and D[i-1] to get C[i] and D[i], respectively. The number of shifts per iteration are given below:

Iteration # 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Left Shifts 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 1

7. Permute the concatenation C[i]D[i] as indicated below. This will yield K[i], which is 48 bits

long. Permuted Choice 2 (PC-2)

8. Loop back to 1.2.3.1 until K[16] has been calculated. Process a 64-bit data block

9. Get a 64-bit data block. If the block is shorter than 64 bits, it should be padded as appropriate for

the application

10. Perform the following permutation on the data block. Initial Permutation (IP)

**Program**

import javax.swing.\*;

import java.security.SecureRandom;

import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.crypto.spec.SecretKeySpec;

import java.util.Random ;

class DES

{

byte[] skey = new byte[1000];

String skeyString;

static byte[] raw;

String inputMessage,encryptedData,decryptedMessage;

public DES()

{

try

{

generateSymmetricKey();

inputMessage=JOptionPane.showInputDialog(null,"Enter message to encrypt");

byte[] ibyte = inputMessage.getBytes();

byte[] ebyte=encrypt(raw, ibyte);

String encryptedData = new String(ebyte);

System.out.println("Encrypted message "+encryptedData);

JOptionPane.showMessageDialog(null,"Encrypted Data "+"\n"+encryptedData);

byte[] dbyte= decrypt(raw,ebyte);

String decryptedMessage = new String(dbyte);

System.out.println("Decrypted message "+decryptedMessage);

JOptionPane.showMessageDialog(null,"Decrypted Data "+"\n"+decryptedMessage);

}

catch(Exception e)

{

System.out.println(e);

}

}

void generateSymmetricKey()

{

try

{

Random r = new Random();

int num = r.nextInt(10000);

String knum = String.valueOf(num);

byte[] knumb = knum.getBytes();

skey=getRawKey(knumb);

skeyString = new String(skey);

System.out.println("DES Symmetric key = "+skeyString);

}

catch(Exception e)

{

System.out.println(e);

}

}

private static byte[] getRawKey(byte[] seed) throws Exception

{

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

SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");

sr.setSeed(seed);

kgen.init(56, sr);

SecretKey skey = kgen.generateKey();

raw = skey.getEncoded();

return raw;

}

private static byte[] encrypt(byte[] raw, byte[] clear) throws Exception

{

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

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

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

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

return encrypted;

}

private static byte[] decrypt(byte[] raw, byte[] encrypted) throws Exception

{

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

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

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

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

return decrypted;

}

public static void main(String args[])

{

DES des = new DES();

}

}

RSA Algorithm

Aim:

To write a Java program to implement RSA algorithm

Algorithm:

1. Generate two large random primes, P and Q, of approximately equal size

2. Compute N = P x Q

3. Compute Z = (P-1) x (Q-1)

4. Choose an integer E, 1 < E < Z, such that GCD (E, Z) = 1

5. Compute the secret exponent D, 1 < D < Z, such that E x D ≡ 1 (mod Z)

6. The public key is (N, E) and the private key is (N, D)

An example of RSA encryption :

1. Select primes P=11, Q=3

2. N = P x Q = 11 x 3 = 33

Z = (P-1) x (Q-1) = 10 x 2 = 20

3. Lets choose E=3

Check GCD(E, P-1) = GCD(3, 10) = 1 (i.e. 3 and 10 have no common factors except 1),

and check GCD(E, Q-1) = GCD(3, 2) = 1, therefore GCD(E, Z) = GCD(3, 20) = 1

4. Compute D such that E x D ≡ 1 (mod Z)

Compute D = E ^-1 mod Z = 3 ^-1 mod 20

Find a value for D such that Z divides ((E x D)-1)

Find D such that 20 divides 3D-1

Simple testing (D = 1, 2, ...) gives D = 7

Check: (E x D)-1 = 3.7 - 1 = 20, which is divisible by Z

5. Public key = (N, E) = (33, 3) and Private key = (N, D) = (33, 7)

Now say we want to encrypt the message m = 7,

Cipher code = M ^E mod N

= 7 ^3 mod 33

= 343 mod 33

= 13

Hence the ciphertext c = 13

To check decryption we compute Message’ = C ^D mod N

= 13 ^7 mod 33

= 7

**Program**

import java.io.DataInputStream;

import java.io.IOException;

import java.math.BigInteger;

import java.util.Random;

public class RSA

{

private BigInteger p;

private BigInteger q;

private BigInteger N;

private BigInteger phi;

private BigInteger e;

private BigInteger d;

private int bitlength = 1024;

private Random r;

public RSA()

{

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 RSA(BigInteger e, BigInteger d, BigInteger N)

{

this.e = e;

this.d = d;

this.N = N;

}

@SuppressWarnings("deprecation")

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

{

RSA rsa = new RSA();

DataInputStream in = new DataInputStream(System.in);

String teststring;

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

teststring = in.readLine();

System.out.println("Encrypting String: " + teststring);

System.out.println("String in Bytes: "+ bytesToString(teststring.getBytes()));

// encrypt

byte[] encrypted = rsa.encrypt(teststring.getBytes());

// decrypt

byte[] decrypted = rsa.decrypt(encrypted);

System.out.println("Decrypting Bytes: " + bytesToString(decrypted));

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

}

private static String bytesToString(byte[] encrypted)

{

String test = "";

for (byte b : encrypted)

{

test += Byte.toString(b);

}

return test;

}

// Encrypt message

public byte[] encrypt(byte[] message)

{

return (new BigInteger(message)).modPow(e, N).toByteArray();

}

// Decrypt message

public byte[] decrypt(byte[] message)

{

return (new BigInteger(message)).modPow(d, N).toByteArray();

}

}

c) Diffiee-Hellman

**Diffie-Hellman Key Exchange**

**Aim:**

**To write a Java program to implement Diffie hellman key exchange algorithm**

**Algorithm:**

**1. Diffie-Hellman key exchange (DH) is a cryptographic protocol that allows two parties that have no prior knowledge of each other to jointly establish a shared secret key**

**2. The algorithm generates a public key and a private key for the client**

**3. Create a KeyPairGenerator Object that generates private/public keys for the DH algorithm, using the getInstance(String algortihm) API method**

**4. Initialize the KeyGenerator so as to generate keys with a 1024-bit length, using the initialize(int keysize) API method**

**5. Create a KeyPair Object , with the genKeyPair() API method, that generates the key pair.**

**6. Create the PrivateKey and PublicKey Objects of the key pair, with the getPrivate() and getPublic() API methods of the KeyPair**

**7. Return for both keys the names of their primary encoded formats, using for both their getformat() ΑPI methods**

**Program**

import java.io.\*;

import java.math.BigInteger;

class Diff

{

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

{

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter prime number:");

BigInteger p=new BigInteger(br.readLine());

System.out.print("Enter primitive root of "+p+":");

BigInteger g=new BigInteger(br.readLine());

System.out.println("Enter value for x less than "+p+":");

BigInteger x=new BigInteger(br.readLine());

BigInteger R1=g.modPow(x,p);

System.out.println("R1="+R1);

System.out.print("Enter value for y less than "+p+":");

BigInteger y=new BigInteger(br.readLine());

BigInteger R2=g.modPow(y,p);

System.out.println("R2="+R2);

BigInteger k1=R2.modPow(x,p);

System.out.println("Key calculated at Alice's side:"+k1);

BigInteger k2=R1.modPow(y,p);

System.out.println("Key calculated at Bob's side:"+k2);

System.out.println("deffie hellman secret key Encryption has Taken");

}

}

**Output**

Enter prime number:

23

Enter primitive root of 23:17

Enter value for x less than 23:

21

R1=19

Enter value for y less than 23:20

R2=16

Key calculated at Alice's side:13

Key calculated at Bob's side:13

deffie hellman secret key Encryption has Taken

d) MD5

**Aim:**

**To write a Java program to implement message digest 5 algorithm**

**Algorithm:**

**1. Append padded bits - The message is padded so that its length is congruent to 448, modulo 512. Means extended to just 64 bits shy of being of 512 bits long. – A single “1” bit is appended to the message, and then “0” bits are appended so that the length in bits equals 448 modulo 512**

**2. Append length - A 64 bit representation of b is appended to the result of the previous step. The resulting message has a length that is an exact multiple of 512 bits**

**3. Initialize MD Buffer - A four-word buffer (A,B,C,D) is used to compute the message digest. – Here each of A,B,C,D, is a 32 bit register. These registers are initialized to the following values in hexadecimal:**

**word A: 01 23 45 67**

**word B: 89 ab cd ef**

**word C: fe dc ba 98**

**word D: 76 54 32 10**

**4. Process message in 16-word blocks – Four auxiliary functions that take as input three 32-bit words and produce as output one 32-bit word. F(X,Y,Z) = XY v not(X) Z G(X,Y,Z) = XZ v Y not(Z) H(X,Y,Z) = X xor Y xor Z I(X,Y,Z) = Y xor (X v not(Z))**

**5. Process message in 16-word blocks cont – if the bits of X, Y, and Z are independent and unbiased, the each bit of F(X,Y,Z), G(X,Y,Z), H(X,Y,Z), and I(X,Y,Z) will be independent and unbiased**

**Program**

import java.io.\*;

import java.io.UnsupportedEncodingException;

import java.math.BigInteger;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

public class Md1 {

public static String getMD5(String input) {

try

{

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

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

BigInteger number = new BigInteger(1, messageDigest);

String hashtext = number.toString(16);

// Now we need to zero pad it if you actually want the full 32 chars.

while (hashtext.length() < 32)

{

hashtext = "0" + hashtext;

}

return hashtext;

}

catch (NoSuchAlgorithmException e)

{

throw new RuntimeException(e);

}

}

public static void main(String[] args) throws NoSuchAlgorithmException

{

try

{

BufferedReader br =new BufferedReader(new InputStreamReader(System.in));

System.out.println(" --------MD5---------");

System.out.println("enter the String to Encrypt:");

String inp=br.readLine();

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ");

System.out.println("Encrypted String: ");

System.out.println(getMD5(inp));

}

catch(Exception e)

{

System.out.println("error");

}

}

}

**Output**

--------MD5---------

enter the String to Encrypt:

sathish kumar

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Encrypted String:

81a2b8d3da8a7db608d676b20bad37b2

**Program**

import java.io.BufferedReader;

import java.io.FileInputStream;

import java.io.InputStreamReader;

import java.io.UnsupportedEncodingException;

import java.math.BigInteger;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

public class Md2

{

public static String getMD5(String input)

{

byte[] source;

try

{

//Get byte according by specified coding.

source = input.getBytes("UTF-8");

}

catch (UnsupportedEncodingException e)

{

source = input.getBytes();

}

String result = null;

char hexDigits[] = {'0', '1', '2', '3', '4', '5', '6', '7','8', '9', 'a', 'b', 'c', 'd', 'e', 'f'};

try

{

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

md.update(source);

//The result should be one 128 integer

byte temp[] = md.digest();

char str[] = new char[16 \* 2];

int k = 0;

for (int i = 0; i < 16; i++) {

byte byte0 = temp[i];

str[k++] = hexDigits[byte0 >>> 4 & 0xf];

str[k++] = hexDigits[byte0 & 0xf];

}

result = new String(str);

}

catch (Exception e)

{

e.printStackTrace();

}

return result;

}

public static void main(String[] args) throws NoSuchAlgorithmException

{

try

{

BufferedReader br =new BufferedReader(new InputStreamReader(System.in));

System.out.println(" --------MD5---------");

System.out.println("enter the String to Encrypt:");

String inp=br.readLine();

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ");

System.out.println("Encrypted String: ");

System.out.println(getMD5(inp));

}

catch(Exception e)

{

System.out.println("error");

}

}

}

**Output**

--------MD5---------

enter the String to Encrypt:

sathish kumar

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Encrypted String:

81a2b8d3da8a7db608d676b20bad37b2

e) SHA-1

**Aim:**

**To write a Java program to implement Secure Hash Algorithm 1**

**Algorithm:**

**1. Append Padding Bits Message is “padded” with a 1 and as many 0’s as necessary to bring the message length to 64 bits fewer than an even multiple of 512**

**2. Append Length 64 bits are appended to the end of the padded message. These bits hold the binary format of 64 bits indicating the length of the original message**

**3. Prepare Processing Function:**

**SHA1 requires 80 processing functions defined as**

**f(t;B,C,D) = (B AND C) OR ((NOT 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)**

**4. Prepare Processing Constants:**

**SHA1 requires 80 processing constant words defined as**

**K(t) = 0x5A827999 ( 0 <= t <= 19)**

**K(t) = 0x6ED9EBA1 (20 <= t <= 39)**

**K(t) = 0x8F1BBCDC (40 <= t <= 59)**

**K(t) = 0xCA62C1D6 (60 <= t <= 79)**

**Program 1**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.UnsupportedEncodingException;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

public class SHA1

{

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

{

BufferedReader userInput = new BufferedReader (new InputStreamReader(System.in));

System.out.println("Enter string:");

String rawString = userInput.readLine();

try

{

System.out.println("SHA1 hash of string: " + SHA2.SHA1(rawString));

}

catch ( NoSuchAlgorithmException | UnsupportedEncodingException e)

{

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

**Program 2**

import java.io.UnsupportedEncodingException;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

public class SHA2

{

private static String convertToHex(byte[] data)

{

StringBuffer buf = new StringBuffer();

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

{

int halfbyte = (data[i] >>> 4) & 0x0F;

int two\_halfs = 0;

do

{

if ((0 <= halfbyte) && (halfbyte <= 9))

buf.append((char) ('0' + halfbyte));

else

buf.append((char) ('a' + (halfbyte - 10)));

halfbyte = data[i] & 0x0F;

}

while(two\_halfs++ < 1);

}

return buf.toString();

}

public static String SHA1(String text)

throws NoSuchAlgorithmException, UnsupportedEncodingException

{

MessageDigest md;

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

byte[] sha1hash = new byte[40];

md.update(text.getBytes("iso-8859-1"), 0, text.length());

sha1hash = md.digest();

return convertToHex(sha1hash);

}

}

**Output**

Enter string:

sathish kumar

SHA1 hash of string: a7ffa6edfd0228bfda755499298779adfb848378

**VIVA QUESTIONS:**

1. **What is DES?**

DES is the Data Encryption Standard, an encryption block cipher defined and endorsed by the U.S. government in 1977 as an official standard; the details can be found in the latest official FIPS (Federal Information Processing Standards) publication concerning DES.

1. **What is Diffie-Hellman?**

It is a method by which a key can be securely shared by two users without any actual exchange.

1. **What is RSA algorithm?**

RSA is short for Rivest-Shamir-Adleman algorithm. It is the most commonly used public key encryption algorithm in use today.

1. **How do you use RSA for both authentication and secrecy?**

RSA is a public key encryption algorithm. The RSA algorithms are based on the mathematical part that it is easy to find and multiply large prime numbers together, but it is extremely difficult to factor their product. For authentication: One can encrypt the hash (MD4/SHA) of the data with a private key. This is known as digital signature. For Secrecy: Secrecy/confidentiality is achieved by encrypting the data with public key and decrypting with private key.

1. **What is SHA-1?**

SHA is a secure hash algorithm developed by National Institute of Standard technology SHA-1 is a type of SHA message digest size of SHA 1 Is 160 bits and message size is < 2^64 it uses block cipher.

**EX. No: 3 Implement the SIGNATURE SCHEME - Digital Signature Standard**

**Date:**

**Aim:**

**To write a Java program to implement digital signature algorithm**

**Algorithm:**

**1. The first part of the DSA algorithm is the public key and private key generation, which can be described as:**

** Choose a prime number q, which is called the prime divisor**

** Choose another primer number p, such that p-1 mod q = 0. p is called the prime modulus**

** Choose an integer g, such that 1 < g < p, g\*\*q mod p = 1 and g = h\*\*((p–1)/q) mod p. q is also called g's multiplicative order modulo p**

** Choose an integer, such that 0 < x < q**

** Compute y as g\*\*x mod p**

** Package the public key as {p,q,g,y}**

** Package the private key as {p,q,g,x}**

**2. The second part of the DSA algorithm are the signature generation and signature verification. To generate a message signature, the sender can follow these steps:**

** Generate the message digest h, using a hash algorithm like SHA1**

** Generate a random number k, such that 0 < k < q**

** Compute r as (g\*\*k mod p) mod q. If r = 0, select a different k**

** Compute i, such that k\*i mod q = 1. i is called modular multiplicative inverse of k modulo q**

** Compute s = i\*(h+r\*x) mod q. If s = 0, select a different k**

** Package the digital signature as {r,s}**

**3. To verify a message signature, receiver of the message and digital signature can follow these**

**steps:**

** Generate the message digest h, using the same hash algorithm**

** Compute w, such that s\*w mod q = 1. w is called the modular multiplicative inverse of s modulo q**

** Compute u1 = h\*w mod q**

** Compute u2 = r\*w mod q**

** Compute v = (((g\*\*u1)\*(y\*\*u2)) mod p) mod q**

**If v == r, the digital signature is valid**

**Program**

import java.util.\*;

import java.math.BigInteger;

class dsaAlg

{

final static BigInteger one = new BigInteger("1");

final static BigInteger zero = new BigInteger("0");

/\* incrementally tries for next prime \*/

public static BigInteger getNextPrime(String ans)

{

BigInteger test = new BigInteger(ans);

while (!test.isProbablePrime(99))

{

test = test.add(one);

}

return test;

}

/\* finds largest prime factor of n \*/

public static BigInteger findQ(BigInteger n)

{

BigInteger start = new BigInteger("2");

while (!n.isProbablePrime(99))

{

while (!((n.mod(start)).equals(zero)))

{

start = start.add(one);

}

n = n.divide(start);

}

return n;

}

/\* finds a generator mod p \*/

public static BigInteger getGen(BigInteger p, BigInteger q,Random r)

{

BigInteger h = new BigInteger(p.bitLength(), r);

h = h.mod(p);

return h.modPow((p.subtract(one)).divide(q), p);

}

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

{

Random randObj = new Random();

/\* establish the global public key components \*/

BigInteger p = getNextPrime("10600");

/\* approximate prime \*/

BigInteger q = findQ(p.subtract(one));

BigInteger g = getGen(p,q,randObj);

/\* public key components \*/

System.out.println("simulation of Digital Signature Algorithm");

System.out.println("global public key components are:");

System.out.println("p is: " + p);

System.out.println("q is: " + q);

System.out.println("g is: " + g);

/\* find the private key \*/

BigInteger x = new BigInteger(q.bitLength(), randObj);

x = x.mod(q);

/\* corresponding public key \*/

BigInteger y = g.modPow(x,p);

/\* random value message \*/

BigInteger k = new BigInteger(q.bitLength(), randObj);

k = k.mod(q);

/\* randomly generated hash value and digital signature \*/

BigInteger r = (g.modPow(k,p)).mod(q);

BigInteger hashVal = new BigInteger(p.bitLength(), randObj);

BigInteger kInv = k.modInverse(q);

BigInteger s = kInv.multiply(hashVal.add(x.multiply(r)));

s = s.mod(q);

System.out.println("secret information are:");

System.out.println("x (private) is: " + x);

System.out.println("k (secret) is: " + k);

System.out.println("y (public) is: " + y);

System.out.println("h (rndhash) is: " + hashVal);

System.out.println("generating digital signature:");

System.out.println("r is : " + r);

System.out.println("s is : " + s);

BigInteger w = s.modInverse(q);

BigInteger u1 = (hashVal.multiply(w)).mod(q);

BigInteger u2 = (r.multiply(w)).mod(q);

BigInteger v = (g.modPow(u1,p)).multiply(y.modPow(u2,p));

v = (v.mod(p)).mod(q);

System.out.println("verifying digital signature (checkpoints):");

System.out.println("w is : " + w);

System.out.println("u1 is : " + u1);

System.out.println("u2 is : " + u2);

System.out.println("v is : " + v);

if (v.equals(r))

{

System.out.println("success: digital signature is verified! " + r);

}

else

{

System.out.println("error: incorrect digital signature");

}

}

}

**Output:**

simulation of Digital Signature Algorithm

global public key components are:

p is: 10601

q is: 53

g is: 7615

secret information are:

x (private) is: 21

k (secret) is: 2

y (public) is: 4224

h (rndhash) is: 2948

generating digital signature:

r is : 13

s is : 47

verifying digital signature (checkpoints):

w is : 44

u1 is : 21

u2 is : 42

v is : 13

success: digital signature is verified! 13

**VIVA QUESTIONS:**

1. **What is the difference between DSA and RSA?**

DSA means Digital Signature Algorithm. The basic difference between DAS and RSA is RSA can encrypt and sign, however DSA is only used for digital signature.

1. **What is Digital Signatures?**

Digital signature is an attachment to an electronic message used for security purpose. It is used to verify the authenticity of the sender.

1. **What is message authentication?**

When the verifier validates the digital signature using public key of a sender, he is assured that signature has been created only by sender who possess the corresponding secret private key and no one else.

1. **What are the services provided by digital certificates?**

Digital certificate provide authentication, nonrepudiation and integrity services.