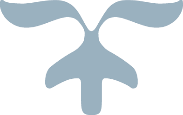


CRYPTOGRAPHY & NETWORK SECURITY

Lab Manual





JNTUK – R20

AUTHOR: RAMYA ASALATHA BUSI

Assoc. Professor, CSE

|  |  |  |
| --- | --- | --- |
| **S.no** | **Program** | **Page no.** |
| **1** | **To Implement and Break Shift Cipher** | **2** |
| **2** | **To Implement Mono – Alphabetic Cipher** | **4** |
| **3** | **To Implement One – Time Pad Cipher** | **6** |
| **4** | **To Implement Message Authentication Codes (MD5)** | **9** |
| **5** | **To Implement Cryptographic Hash Function (SHA-256)** | **11** |
| **6** | **To Implement Symmetric Encryption Cipher DES** | **13** |
| **7** | **To Implement Symmetric Encryption Cipher AES** | **15** |
| **8** | **To Implement Diffie – Hellman Key Establishment** | **17** |
| **9** | **To Implement Public – Key Cryptosystems (RSA)** | **20** |
| **10** | **To Implement Digital Signatures (DSA)** | **22** |

# Aim: Program to break Shift Cipher Program:

import java.util.\*; class CaesarCipher

{

public static void main(String args[])

{

Scanner sc=new Scanner(System.in); int shift,i,n,p,key;

String str; String str1="";

System.out.println("Enter the Plain Text"); str=sc.nextLine();

str=str.toLowerCase(); n=str.length();

char ch1[]=str.toCharArray(); char ch4;

System.out.println("Enter the value by which each letter of the string is to be shifted"); shift=sc.nextInt();

System.out.println(); System.out.println("Encrypted text is:"); for(i=0;i<n;i++)

{

if(Character.isLetter(ch1[i]))

{

ch4=(char)(((int)ch1[i]+shift-97)%26+97); str1=str1+ch4;

}

else if(ch1[i]==' ')

{

str1=str1+ch1[i];

}

}

System.out.println(str1); System.out.println("Cipher Text:"+str1); n=str1.length();

char ch2[]=str1.toCharArray(); char ch3; System.out.println();

System.out.println("Possible Plain text is"); str1="";

for(key=26;key>=1;key--)

{

for(i=0;i<n;i++)

{

if(Character.isLetter(ch2[i]))

{

ch3=(char)(((int)ch2[i]+key-97)%26+97); str1=str1+ch3;

}

else if(ch2[i]==' ')

{

str1=str1+ch2[i];

}

}

p=26-key;

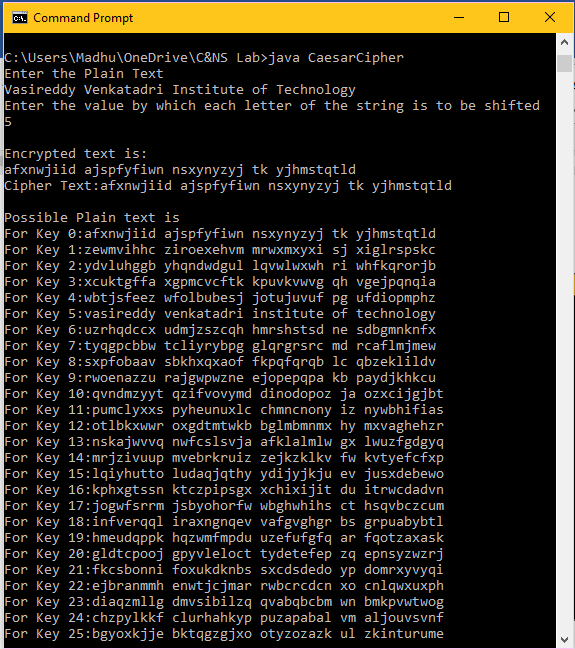
System.out.println("For Key "+p+":"+str1); str1="";

}

}

}

# Output:



1. **Aim: Program to break Mono-alphabetic cipher Program:**

import java.util.Scanner;

public class MonoalphabeticCipher

{

public static char p[] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',

'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z' };

public static char ch[] = { 'Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O',

'P', 'A', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L', 'Z', 'X', 'C', 'V', 'B', 'N', 'M' };

static String str;

public static String doEncryption(String s)

{

char c[] = new char[(s.length())]; for (int i = 0; i < s.length(); i++)

{

for (int j = 0; j < 26; j++)

{

if (p[j] == s.charAt(i))

{

c[i] = ch[j]; break;

}

}

}

return (new String(c));

}

public static String doDecryption(String s)

{

char p1[] = new char[(s.length())]; for (int i = 0; i < s.length(); i++)

{

for (int j = 0; j < 26; j++)

{

if (ch[j] == s.charAt(i))

{

p1[i] = p[j]; break;

}

}

}

return (new String(p1));

}

public static void main(String args[])

{

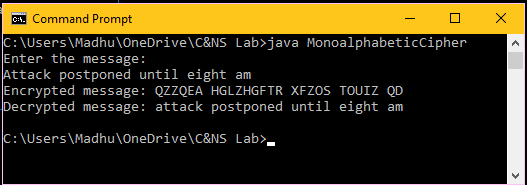
Scanner sc = new Scanner(System.in); System.out.println("Enter the message: "); str=sc.next();

String en = doEncryption(str.toLowerCase()); System.out.println("Encrypted message: " + en); System.out.println("Decrypted message: " + doDecryption(en)); sc.close();

}

}

# Output:



1. **Aim: Program to implement One-time pad Program:**

import java.util.\*; class msg{

int a=97;

char all[]=new char[27]; msg()

{

for(int i=0;i<26;i++)

{

all[i]=(char)a; a++;

}

}

int Ipos(char c)

{

int i=0; for(;i<26;i++)

{

if(all[i]==c)

{

break;

}

}

return i;

}

char Cpos(int c)

{

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

{

}

return all[i];

}

}

class OneTimePadCipherImplementation{ String Encryption(String plaintext,String key)

{

plaintext=plaintext.toLowerCase(); msg m1=new msg();

int pt[]=new int[plaintext.length()]; int k[]=new int[key.length()];

int ct[]=new int[plaintext.length()]; for(int i=0;i < plaintext.length();i++)

{

pt[i]=m1.Ipos(plaintext.charAt(i));

}

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

{

k[i]=m1.Ipos(key.charAt(i));

}

int j=0;

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

{

ct[i]=pt[i]+k[j]; j++;

if(j==key.length()) j=0;

if(ct[i]>26)

ct[i]=ct[i]%26;

}

String cipher="";

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

{

cipher+=m1.Cpos(ct[i]);

}

return cipher;

}

String Decryption(String ciphertext,String key)

{

String plaintext=""; msg m1=new msg();

int pt[]=new int[ciphertext.length()]; int k[]=new int[key.length()];

int ct[]=new int[ciphertext.length()]; for(int i=0;i < ciphertext.length();i++)

{

ct[i]=m1.Ipos(ciphertext.charAt(i));

}

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

{

k[i]=m1.Ipos(key.charAt(i));

}

int j=0;

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

{

pt[i]=ct[i]-k[j]; j++;

if(j==key.length()) j=0;

if(pt[i] < 0) pt[i]+=26;

}

String cipher="";

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

{

plaintext+=m1.Cpos(pt[i]);

}

return plaintext;

}

}

class OneTimePad{

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

{

String plaintext,key;

Scanner scn=new Scanner(System.in); System.out.println("Enter plaintext:"); plaintext=scn.nextLine(); System.out.println("Enter key:"); key=scn.nextLine();

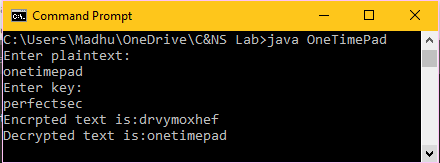
OneTimePadCipherImplementation OneTimePad=new OneTimePadCipherImplementation();

String ciphertext=OneTimePad.Encryption(plaintext,key); System.out.println("Encrpted text is:"+ciphertext); System.out.println("Decrypted text is:"+OneTimePad.Decryption(ciphertext,key));

}

}

# Output:



1. **Aim: Program to implement Message authentication codes (MD5) Program:**

import java.math.BigInteger; import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

// Java program to calculate MD5 hash value public class MD5 {

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

{

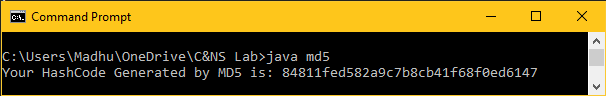
String s = "VVIT";

System.out.println("Your HashCode Generated by MD5 is: " + getMd5(s));

}

}

# Output:



1. **Aim: Program to implement Cryptographic Hash Function (SHA-256) Program:**

import java.math.BigInteger; import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

// Java program to calculate MD5 hash value public class SHA {

public static String getSHA(String input)

{

try {

// Static getInstance method is called with hashing MD5 MessageDigest hash = MessageDigest.getInstance("SHA-256");

// digest() method is called to calculate message digest

// of an input digest() return array of byte

byte[] messageDigest = hash.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

{

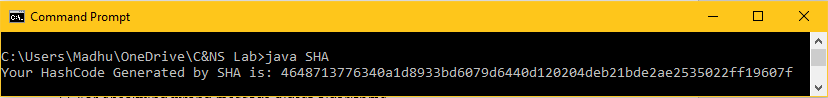
String s = "VVIT";

System.out.println("Your HashCode Generated by SHA is: " + getSHA(s));

}

}

# Output:



1. **Aim: Program to implement DES Symmetric Encryption Program:**

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

class DesEncrypter { Cipher ecipher; Cipher dcipher;

DesEncrypter(SecretKey key) throws Exception { ecipher = Cipher.getInstance("DES");

dcipher = Cipher.getInstance("DES"); ecipher.init(Cipher.ENCRYPT\_MODE, key); dcipher.init(Cipher.DECRYPT\_MODE, key);

}

public String encrypt(String str) throws Exception {

// Encode the string into bytes using utf-8 byte[] utf8 = str.getBytes("UTF8");

// Encrypt

byte[] enc = ecipher.doFinal(utf8);

// Encode bytes to base64 to get a string

//return new sun.misc.BASE64Encoder().encode(enc); return Base64.getEncoder().encodeToString(enc);

}

public String decrypt(String str) throws Exception {

// Decode base64 to get bytes

//byte[] dec = new sun.misc.BASE64Decoder().decodeBuffer(str); byte[] dec = Base64.getDecoder().decode(str);

byte[] utf8 = dcipher.doFinal(dec);

// Decode using utf-8

return new String(utf8, "UTF8");

}

}

public class DES {

public static void main(String[] argv) throws Exception { SecretKey key = KeyGenerator.getInstance("DES").generateKey(); DesEncrypter encrypter = new DesEncrypter(key);

String encrypted = encrypter.encrypt("Don't tell anybody!"); System.out.println(encrypted);

String decrypted = encrypter.decrypt(encrypted); System.out.println(decrypted);

}

}

# Output:



1. **Aim: Program to implement AES Symmetric Encryption Program:**

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

class AesEncrypter { Cipher ecipher;

Cipher dcipher;

AesEncrypter(SecretKey key) throws Exception { ecipher = Cipher.getInstance("AES");

dcipher = Cipher.getInstance("AES"); ecipher.init(Cipher.ENCRYPT\_MODE, key); dcipher.init(Cipher.DECRYPT\_MODE, key);

}

public String encrypt(String str) throws Exception {

// Encode the string into bytes using utf-8 byte[] utf8 = str.getBytes("UTF8");

// Encrypt

byte[] enc = ecipher.doFinal(utf8);

// Encode bytes to base64 to get a string

//return new sun.misc.BASE64Encoder().encode(enc); return Base64.getEncoder().encodeToString(enc);

}

public String decrypt(String str) throws Exception {

// Decode base64 to get bytes

//byte[] dec = new sun.misc.BASE64Decoder().decodeBuffer(str); byte[] dec = Base64.getDecoder().decode(str);

byte[] utf8 = dcipher.doFinal(dec);

// Decode using utf-8

return new String(utf8, "UTF8");

}

}

public class AES {

public static void main(String[] argv) throws Exception { SecretKey key = KeyGenerator.getInstance("AES").generateKey(); AesEncrypter encrypter = new AesEncrypter(key);

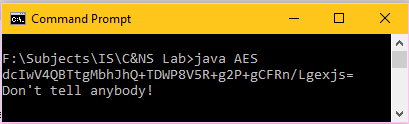
String encrypted = encrypter.encrypt("Don't tell anybody!"); System.out.println(encrypted);

String decrypted = encrypter.decrypt(encrypted); System.out.println(decrypted);

}

}

# Output:



1. **Aim: Program to implement Diffie – Hellman Key Establishment Program:**

**DHServer.java**

import java.net.\*; import java.io.\*;

public class DHServer {

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

{

"...");

try {

int port = 8088;

// Server Key int b = 3;

// Client p, g, and key

double clientP, clientG, clientA, B, Bdash; String Bstr;

// Established the Connection

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

Socket server = serverSocket.accept();

System.out.println("Just connected to " + server.getRemoteSocketAddress());

// Server's Private Key

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

// Accepts the data from client

DataInputStream in = new DataInputStream(server.getInputStream());

clientP = Integer.parseInt(in.readUTF()); // to accept p System.out.println("From Client : P = " + clientP);

clientG = Integer.parseInt(in.readUTF()); // to accept g System.out.println("From Client : G = " + clientG);

clientA = Double.parseDouble(in.readUTF()); // to accept A System.out.println("From Client : Public Key = " + clientA);

B = ((Math.pow(clientG, b)) % clientP); // calculation of B Bstr = Double.toString(B);

// Sends data to client

// Value of B

OutputStream outToclient = server.getOutputStream(); DataOutputStream out = new DataOutputStream(outToclient);

out.writeUTF(Bstr); // Sending B

Bdash = ((Math.pow(clientA, b)) % clientP); // calculation of Bdash System.out.println("Secret Key to perform Symmetric Encryption = "

+ Bdash);

server.close();

}

catch (SocketTimeoutException s) { System.out.println("Socket timed out!");

}

catch (IOException e) {

}

}

}

**DHClient.java**

import java.net.\*; import java.io.\*;

public class DHClient {

public static void main(String[] args)

{

try {

String pstr, gstr, Astr;

String serverName = "localhost"; int port = 8088;

// Declare p, g, and Key of client int p = 23;

int g = 9; int a = 4;

double Adash, serverB;

// Established the connection System.out.println("Connecting to " + serverName

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

+ client.getRemoteSocketAddress());

// Sends the data to client

OutputStream outToServer = client.getOutputStream(); DataOutputStream out = new DataOutputStream(outToServer);

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

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

double A = ((Math.pow(g, a)) % p); // calculation of A Astr = Double.toString(A);

out.writeUTF(Astr); // Sending A

// Client's Private Key

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

// Accepts the data

DataInputStream in = new DataInputStream(client.getInputStream());

serverB = Double.parseDouble(in.readUTF()); System.out.println("From Server : Public Key = " + serverB);

Adash = ((Math.pow(serverB, a)) % p); // calculation of Adash System.out.println("Secret Key to perform Symmetric Encryption = "

+ Adash);

client.close();

}

catch (Exception e) {

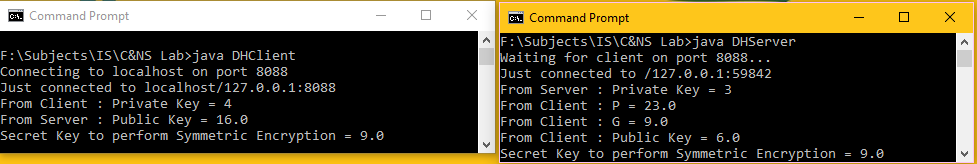
e.printStackTrace();

}

}

}

# Output:



1. **Aim: Program to implement Public Key Cryptosystems (RSA) Program:**

import java.math.BigInteger;

import java.security.SecureRandom;

public class RSADemo {

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

private final static SecureRandom random = new SecureRandom();

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

// generate an N-bit (roughly) public and private key RSADemo(int N) {

BigInteger p = BigInteger.probablePrime(N/2, random); BigInteger q = BigInteger.probablePrime(N/2, random); BigInteger phi = (p.subtract(one)).multiply(q.subtract(one)); System.out.println("prime p = " + p); System.out.println("prime q = " + q);

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

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

}

BigInteger encrypt(BigInteger message) {

return message.modPow(publicKey, modulus);

}

BigInteger decrypt(BigInteger encrypted) {

return encrypted.modPow(privateKey, modulus);

}

public String toString() { String s = "";

s += "public = " + publicKey + "\n"; s += "private = " + privateKey + "\n"; s += "modulus = " + modulus; return s;

}

public static void main(String[] args) { int N = Integer.parseInt(args[0]); RSADemo key = new RSADemo(N); System.out.println(key);

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

//// create message by converting string to integer

// String s = "test";

// byte[] bytes = s.getBytes();

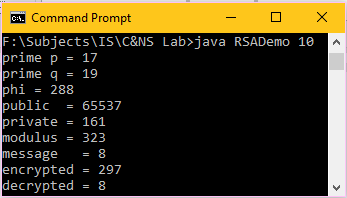
// BigInteger message = new BigInteger(bytes);

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

}

}

# Output:



1. **Aim: Program to Implement Digital Signatures (DSS) Program:**

import java.security.KeyPair;

import java.security.KeyPairGenerator; import java.security.PrivateKey; import java.security.PublicKey;

import java.security.Signature; import java.util.Scanner; import java.io.\*;

public class CreatingDigitalSignature {

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

//Accepting text from user

Scanner sc = new Scanner(System.in); System.out.println("Enter some text"); String msg = sc.nextLine();

//Creating KeyPair generator object

KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("DSA");

//Initializing the key pair generator keyPairGen.initialize(2048);

//Generate the pair of keys

KeyPair pair = keyPairGen.generateKeyPair();

//Getting the private key from the key pair PrivateKey privKey = pair.getPrivate(); PublicKey pubKey = pair.getPublic(); System.out.println(privKey); System.out.println(pubKey);

//Creating a Signature object

Signature sign = Signature.getInstance("SHA256withDSA");

//Initialize the signature sign.initSign(privKey);

byte[] bytes = "msg".getBytes();

//Adding data to the signature sign.update(bytes);

//Calculating the signature byte[] signature = sign.sign();

//Printing Signature to console for(int i=0;i<signature.length;i++)

System.out.print(" "+signature[i]); System.out.println();

//Initializing the signature verification sign.initVerify(pair.getPublic()); sign.update(bytes);

//Verifying the signature

boolean bool = sign.verify(signature);

if(bool) {

System.out.println("Signature verified");

} else {

System.out.println("Signature failed");

}

}

}

# Output:

