Gokaraju Rangaraju Institute of Engineering and Technology

(Autonomous)

Department of Artificial Intelligence and Machine Learning

CRYPTOGRAPHY AND NETWORK SECURITY LAB

LAB MANUAL

Program: IV B.Tech I Semester

Course Code: GR20A34054

Academic Year: 2023-2024

Faculty Name: Ms. Manu Hajari / Mr. N. Devendher

Designation: Assistant Professor

Syllabus

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY
CRYPTOGRAPHY AND NETWORK SECURITY LAB

Course Code: GR20A34054 **L/T/P/C:0/0/4/2**

IV Year I Semester

Course Objectives:

1. Explain different types of ciphers used for encryption and decryption.
2. Demonstrate on symmetric encryption algorithms.
3. Demonstrate on asymmetric encryption algorithms.
4. Experiment on Hash algorithms.
5. Illustrate programs related to digital certificates and digital signatures.

Course Outcomes:

1. Use the concepts of different ciphers for encryption and decryption.
2. Implement symmetric encryption algorithms.
3. Examine asymmetric encryption algorithms.
4. Interpret hash algorithms and their functionalities.
5. Solve the problems on digital signatures and digital certificates.

TASK 1:
Write a Java program to perform encryption and decryption using the following algorithms.

a. Ceaser cipher b. Substitution cipher c. Hill Cipher

TASK 2:
Write a C/ JAVA program to implement the DES algorithm.

TASK 3:
Write a C/JAVA program to implement the Blowfish algorithm.

TASK 4:
Write a C/JAVA program to implement the AES algorithm .

TASK 5:
Write the RC4 logic in Java.

TASK 6:
Implement DES-2 and DES-3 using Java cryptography package.

TASK 7:
Write a Java program to implement RSA algorithm.

TASK 8:
Implement the Diffie-Hellman Key Exchange mechanism

TASK 9:
Calculate the message digest of a text using the SHA-1 algorithm in JAVA.

TASK 10:
Calculate the message digest of a text using the MD5 algorithm in JAVA.

TASK 11:
Explore the Java classes related to digital certificates.

TASK 12:
Write a program in java, which performs a digital signature on a given text.

Text Books:

1. Network Security Essentials (Applications and Standards) William StallingsPearson Education.
2. Fundamentals of Network security by Eric Maiwald (Dreamtechpress)

References:

1. Introduction to Cryptography, Buchmann,Springer.
2. Cryptography and network security, Third Edition.

INDEX

TASK 1

Task 1(a): Write a Java program to perform encryption and decryption using the algorithm Ceaser Cipher.

Aim: To write a Program on encryption and decryption using ceaser cipher.

Description:

To encrypt a message with a ceaser cipher,the letter in each message is changed using a simple rule : shift by three. Each letter is replaced by the letter three letters ahead in the alphabet .A becomes D,B becomes E,and so on.for the last letters , we can think of the alphabet as a circle and "wrap around".W becomes Z,X becomes A,Y becomes B and Z becomes C.To change a message back, each letter is replaced by the one three before it.

Program:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;
public class CeaserCipher
{
        static Scanner sc = new Scanner(System.in);
        static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
public static void main(String[] args) throws IOException
{
                // TODO code application logic here
                System.out.print("Enter any String: ");
                String str = br.readLine();
                System.out.print("\nEnter the Key: ");
                int key = sc.nextInt();
                String encrypted = encrypt(str, key);
                System.out.println("\nEncrypted String is: " + encrypted);
                String decrypted = decrypt(encrypted, key);
                System.out.println("\nDecrypted String is: " + decrypted);
                System.out.println("\n");
        }
public static String encrypt(String str, int key)
{
String encrypted = "";
                for (int i = 0; i < str.length(); i++)
{
                        int c = str.charAt(i);
                        if (Character.isUpperCase(c))
{
                                c = c + (key % 26);
                                if (c > 'Z')
                                        c = c - 26;
                        }
```

```
else if (Character.isLowerCase(c))
{
                                        c = c + (key % 26);
                                        if (c > 'z')
                                                c = c - 26;
                                }
                                encrypted += (char) c;
                        }
                        return encrypted;
                }
                public static String decrypt(String str, int key)
{
                        String decrypted = "";
                        for (int i = 0; i < str.length(); i++)
{
                                int c = str.charAt(i);
                                if (Character.isUpperCase(c))
{
                                        c = c - (key % 26);
                                        if (c < 'A')
                                                c = c + 26;
                                }
else if (Character.isLowerCase(c))
{
                                        c = c - (key % 26);
                                        if (c < 'a')
                                                c = c + 26;
                                }
                                decrypted += (char) c;
                        }
                        return decrypted;
                }
}
```

OUTPUT:
Enter any String: HELLO

Enter the Key: 3

Encrypted String is: KHOOR

Decrypted String is: HELLO

Task 1(b): Write a Java program to perform encryption and decryption using the algorithm Substitution Cipher.

Aim: To write a Java Program using substitution cipher.

Program:

```
import java.io.*;
import java.util.*;
public class SubstitutionCipher
{
static Scanner sc = new Scanner(System.in);
static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
public static void main(String[] args) throws IOException
{
String encrypt = encrypt();
String decrypt = decrypt(encrypt);
System.out.println("The encrypted data is: " +encrypt);
System.out.println("The decrypted data is: " +decrypt);
}
public static String encrypt()throws IOException
{
        String encrypt = "";
        String a = "abcdefghijklmnopqrstuvwxyz";
        String b = "zyxwvutsrqponmlkjihgfedcba";
        System.out.print("Enter any string: ");
        String str = br.readLine();
        char c;
        for(int i=0;i<str.length();i++)
        {
        c = str.charAt(i);
        int j = a.indexOf(c);
        encrypt = encrypt+b.charAt(j);
        }
        return encrypt;
}
```

```
public static String decrypt(String encrypt)
{
        String a = "abcdefghijklmnopqrstuvwxyz";
        String b = "zyxwvutsrqponmlkjihgfedcba";
        String decrypt = "";
        char c;
        for(int i=0;i<encrypt.length();i++)
        {
        c = encrypt.charAt(i);
        int j = a.indexOf(c);
        decrypt = decrypt+b.charAt(j);
        }
        return decrypt;
}
}
```

OUTPUT:

Enter any string: hello

The encrypted data is: svool

The decrypted data is: hello

Task 1(c): Write a Java program to perform encryption and decryption using the algorithm Hill Cipher.

Aim: To write a Java Program using Hill cipher.

Description:

Each letter is represented by a number modulo*26.Often the simple schemeA=0,B= 1... Z =* a *25, is used, but this is not an essential feature of the cipher. To encrypt a message, each block of* n *letters is multiplied by an invertible* n × n matrix, against modulus 26. To decrypt the message, each block is multiplied by the inverse of the m trixused for encryption. The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible *n* × *n* matrices (modulo 26).

Program:

```
package com.islab;
import java.util.*;


class Basic {
   String allChar = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";


   int indexOfChar(char c) {
      for (int i = 0; i < allChar.length(); i++) {
         if (allChar.charAt(i) == c)
            return i;
      }
      return -1;
   }


   char charAtIndex(int pos) {
      return allChar.charAt(pos);
   }
}


class Hill {
   Basic b1 = new Basic();
   int block;
   int key[][];
```

```
Hill(int block) {
    this.block = block;
    this.key = new int[block][block];
}

void keyInsert() throws Exception {
    Scanner scn = new Scanner(System.in);
    System.out.println("Enter key Matrix");
    for (int i = 0; i < block; i++) {
        for (int j = 0; j < block; j++) {
            key[i][j] = scn.nextInt();
        }
    }
}

void KeyInverseInsert() throws Exception {
    Scanner scn = new Scanner(System.in);
    System.out.println("Enter key Inverse Matrix:");
    for (int i = 0; i < block; i++) {
        for (int j = 0; j < block; j++) {
            key[i][j] = scn.nextInt();
        }
    }
}

String encryptBlock(String plain) throws Exception {
    plain = plain.toUpperCase();
    int a[][] = new int[block][1], sum = 0;
    int cipherMatrix[][] = new int[block][1];
    String cipher = "";

    for (int i = 0; i < block; i++) {
        a[i][0] = b1.indexOfChar(plain.charAt(i));
    }
```

```
    for (int i = 0; i < block; i++) {
        for (int j = 0; j < 1; j++) {
            for (int k = 0; k < block; k++) {
                sum = sum + key[i][k] * a[k][j];
            }
            cipherMatrix[i][j] = sum % 26;
            sum = 0;
        }
    }

    for (int i = 0; i < block; i++) {
        cipher += b1.charAtIndex(cipherMatrix[i][0]);
    }

    return cipher;
}

String encrypt(String plainText) throws Exception {
    String cipherText = "";
    keyInsert();
    plainText = plainText.toUpperCase();
    int len = plainText.length();

    while (len % block != 0) {
        plainText += "X";
        len = plainText.length();
    }

    for (int i = 0; i < len; i = i + block) {
        cipherText += encryptBlock(plainText.substring(i, i + block));
        cipherText += " ";
    }
```

```
    return cipherText;
}

String decryptBl(String cipher) throws Exception {
    cipher = cipher.toUpperCase();
    int a[][] = new int[block][1], sum = 0;
    int plainMatrix[][] = new int[block][1];
    String plain = "";

    for (int i = 0; i < block; i++) {
        a[i][0] = b1.indexOfChar(cipher.charAt(i));
    }

    for (int i = 0; i < block; i++) {
        for (int j = 0; j < 1; j++) {
            for (int k = 0; k < block; k++) {
                sum = sum + key[i][k] * a[k][j];
            }
            while (sum < 0) {
                sum += 26;
            }
            plainMatrix[i][j] = sum;
            sum = 0;
        }
    }

    for (int i = 0; i < block; i++) {
        plain += b1.charAtIndex(plainMatrix[i][0]);
    }

    return plain;
}

String Decrypt(String cipherText) throws Exception {
```

```
        String plainText = "";
        KeyInverseInsert();
        cipherText = cipherText.replaceAll(" ", "");
        cipherText = cipherText.toUpperCase();
        int len = cipherText.length();

        for (int i = 0; i < len; i = i + block) {
            plainText += decryptBl(cipherText.substring(i, i + block));
            plainText += " ";
        }

        return plainText;
    }
}

class HillCipher {
    public static void main(String args[]) throws Exception {
        String plainText, cipherText;
        int block;
        Scanner scn = new Scanner(System.in);
        System.out.println("Enter plain-text:");
        plainText = scn.nextLine();
        System.out.println("Enter block size of matrix:");
        block = scn.nextInt();
        Hill hill = new Hill(block);
        plainText = plainText.replaceAll(" ", "");
        cipherText = hill.encrypt(plainText);
        System.out.println("Encrypted Text is:\n" + cipherText);
        String decryptedText = hill.Decrypt(cipherText);
        System.out.println("Decrypted Text is:\n" + decryptedText);
    }
}
```

OUTPUT:

Enter plain-text:

meet

Enter block size of matrix:

2

Enter key Matrix

3 1

5 2

Encrypted Text is:

OQ FG

Enter key Inverse Matrix:

2 -1

-5 3

Decrypted Text is:

ME ET

TASK 2

Write a C/JAVA Program to implement the DES algorithm Logic.

Aim: To write a Java Program on DES algorithm

Description:

DES is a symmetric encryption system that uses 64-bit blocks, 8 bits of which are used for parity checks. The key therefore has a "useful" length of 56 bits, which means that only 56 bits are actually used in the algorithm. The algorithm involves carrying out combinations, substitutions and permutations between the text to be encrypted and the key, while making sure the operations can be performed in both directions. The key is ciphered on 64 bits and made of 16 blocks of 4 bits, generally denoted k_1 to k_{16}. Given that "only" 56 bits are actually used for encrypting, there can be 2^{56} differentkeys.

The main parts of the algorithm are as follows:

Fractioning of the text into 64-bitblocks
Initial permutation ofblocks
Breakdown of the blocks into two parts: left and right, named L andR
Permutation and substitution steps repeated 16times
Re-joining of the left and right parts then inverse initialpermutation

Program:

```
package com.islab;

package javaprogram;

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 DES1 {

byte[] skey = new byte[1000];

String skeyString;

static byte[] raw;

String inputMessage,encryptedData,decryptedMessage;


public DES1() {

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");
```

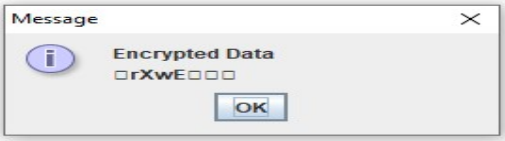
```
SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
sr.setSeed(seed);
kgen.init(56, sr);
SecretKey skey = kgen.generateKey();
raw = skey.getEncoded();
return raw;
}
private static byte[] encrypt(byte[] raw, byte[] clear) throws Exception {
SecretKeySpec skeySpec = new SecretKeySpec(raw, "DES");
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[]) {
DES1 des = new DES1();
}
}
```

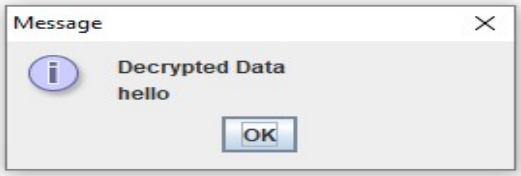
Output:

```
run:
DES Symmetric key = □Qu0□□k□
```


```
run:
DES Symmetric key = ␣□,□R□m
Encrypted message □rXwE□□□
```




```
run:
DES Symmetric key = □□v,□u□
Encrypted message S□□h]□
Decrypted message hello
```



TASK 3

Write a C/JAVA Program to implement the Blowfish algorithm logic

Aim: To write a Program to implement Blowfish algorithm logic.

Program:

```
package com.islab;
import java.io.*;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.security.Key;
import javax.crypto.Cipher;
import javax.crypto.CipherOutputStream;
import javax.crypto.KeyGenerator;
import sun.misc.BASE64Encoder;
public class BlowFish {
public static void main(String[] args) throws Exception {
 // TODO code application logic here
KeyGenerator keyGenerator = KeyGenerator.getInstance("Blowfish");
keyGenerator.init(128);
Key secretKey = keyGenerator.generateKey();
Cipher cipherOut = Cipher.getInstance("Blowfish/CFB/NoPadding");
cipherOut.init(Cipher.ENCRYPT_MODE, secretKey);
BASE64Encoder encoder = new BASE64Encoder();
byte iv[] = cipherOut.getIV();
if (iv != null) {
System.out.println("Initialization Vector of the Cipher: " + encoder.encode(iv));
}
FileInputStream fin = new FileInputStream("D:/ISLABPrograms/inputFile.txt");
FileOutputStream fout = new FileOutputStream("D:/ISLABPrograms/outputFile.txt");
CipherOutputStream cout = new CipherOutputStream(fout, cipherOut);
int input = 0;
while ((input = fin.read()) != -1)
{
cout.write(input);
}
fin.close();
cout.close();
}
}
```

OUTPUT:

```
Initialization Vector of the Cipher: h4S9qmjXhRU=
```

TASK 4

Write a C/JAVA Program to implement the AES algorithm logic.

Aim: To implement AES Algorithm logic.

Program:

```
package com.islab;
import java.io.*;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.xml.bind.DatatypeConverter;

/**
 * This example program shows how AES encryption and decryption can be done in java
 * Please note that secret key and encrypted text is unreadable binary and hence
 * in the following program we display it in hexadecimal format of the
 * underlying bytes.
 */

public class AESEncryption
{
public static void main(String[] args) throws Exception
        {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println(" Enter Text to be Encrypted::");
        String plainText = br.readLine();
        SecretKey secKey = getSecretEncryptionKey();
        byte[] cipherText = encryptText(plainText, secKey);
        String decryptedText = decryptText(cipherText, secKey);
        System.out.println("Original Text:" + plainText);
        System.out.println("AES Key (Hex Form):"+ bytesToHex(secKey.getEncoded()));
        System.out.println("Encrypted Text (Hex Form):"+ bytesToHex(cipherText));
        System.out.println("Descrypted Text:" + decryptedText);
        }
       /**
         * gets the  AES encryption key. In your actual programs, this should be
         * safely stored.
         * @return
         * @throws Exception
         */
        public static SecretKey getSecretEncryptionKey() throws Exception
        {
                KeyGenerator generator = KeyGenerator.getInstance("AES");
                generator.init(128); // The AES key size in number of bits
                SecretKey secKey = generator.generateKey();
                return secKey;

        }
```

```
/**
	* Encrypts plainText in AES using the secret key
	* @param plainText
	* @param secKey
	* @return
	* @throws Exception
	 */

public static byte[] encryptText(String plainText, SecretKey secKey) throws Exception
{
// AES defaults to AES/ECB/PKCS5Padding in Java 7
Cipher aesCipher = Cipher.getInstance("AES");
aesCipher.init(Cipher.ENCRYPT_MODE, secKey);
byte[] byteCipherText = aesCipher.doFinal(plainText.getBytes());
return byteCipherText;
}
/**
	 * Decrypts encrypted byte array using the key used for encryption.
	 * @param byteCipherText
	 * @param secKey
	 * @return
	 * @throws Exception
	 */
public static String decryptText(byte[] byteCipherText, SecretKey secKey) throws Exception
{
// AES defaults to AES/ECB/PKCS5Padding in Java 7
		Cipher aesCipher = Cipher.getInstance("AES");
		aesCipher.init(Cipher.DECRYPT_MODE, secKey);
		byte[] bytePlainText = aesCipher.doFinal(byteCipherText);
		return new String(bytePlainText);
}
	private static String bytesToHex(byte[] hash)
	{
	return DatatypeConverter.printHexBinary(hash);
	}
}
```

OUTPUT:

```
Enter Text to be Encrypted::
play
Original Text: play
AES Key (Hex Form):BFB4BAB28E3A16EC9BDB559614B6093F
Encrypted Text (Hex Form):7E508E590685397531D5E95B1905EE09
Descrypted Text:play
```

TASK 5

Write the RC4 logic in JAVA

Aim: To implement RC4 LOGIC

Program:

```
package com.islab;
import java.io.*;
class RCA4
{
public static void main(String args[]) throws IOException
{
int temp = 0;
String ptext;
String key;
int s[] = new int[256];
int k[] = new int[256];
DataInputStream in = new DataInputStream(System.in);
System.out.println("\n ENTER PLAIN TEXT\t");
ptext = in.readLine();
System.out.println("\n\nENTER KEY TEXT\t\t");
key = in.readLine();
char ptextc[] = ptext.toCharArray();
char keyc[] = key.toCharArray();
int cipher[] = new int[ptext.length()];
int decrypt[] = new int[ptext.length()];
int ptexti[] = new int[ptext.length()];
int keyi[] = new int[key.length()];
        for (int i = 0; i < ptext.length(); i++)
        {
        ptexti[i] = (int) ptextc[i];
        }
        for (int i = 0; i < key.length(); i++)
        {
        keyi[i] = (int) keyc[i];
        }
        for (int i = 0; i < 255; i++)
        {
        s[i] = i;
        k[i] = keyi[i % key.length()];
        }
        int j = 0;
        for (int i = 0; i < 255; i++)
        {
        j = (j + s[i] + k[i]) % 256;
        temp = s[i];
        s[i] = s[j];
        s[j] = temp;
        }
```

```
        int i = 0;
        j = 0;
        int z = 0;
        for (int l = 0; l < ptext.length(); l++)
        {
        i = (l + 1) % 256;
        j = (j + s[i]) % 256;
        temp = s[i];
        s[i] = s[j];
        s[j] = temp;
        z = s[(s[i] + s[j]) % 256];
        cipher[l] = z ^ ptexti[l];
        decrypt[l] = z ^ cipher[l];
        }
        System.out.println("\n\nENCRYPTED:\t\t");
        display(cipher);
        System.out.println("\n\nDECRYPTED:\t\t");
        display(decrypt);
        }
        static void display(int disp[])
        {
        char convert[] = new char[disp.length];
        for (int l = 0; l < disp.length; l++)
        {
        convert[l] = (char) disp[l];
        System.out.print(convert[l]);
        }
}

}
```

Output:

ENTER PLAIN TEXT
hello

ENTER KEY TEXT
krishna

ENCRYPTED:
Iel1|

DECRYPTED:
hello

TASK 6

Implement DES-2 and DES-3 using Java cryptography package

Aim: To implement DES-2 and DES-3 using Java cryptography package

Description:

DES is a symmetric encryption system that uses 64-bit blocks, 8 bits of which are used for parity checks. The key therefore has a "useful" length of 56 bits, which means that only 56 bits are actually used in the algorithm. The algorithm involves carrying out combinations, substitutions and permutations between the text to be encrypted and the key, while making sure the operations can be performed in both directions. The key is ciphered on 64 bits and made of 16 blocks of 4 bits, generally denoted k_1 to k_{16}. Given that "only" 56 bits are actually used for encrypting, there can be 2^{56} differentkeys.

The main parts of the algorithm are as follows:

Fractioning of the text into 64-bitblocks
Initial permutation ofblocks
Breakdown of the blocks into two parts: left and right, named L andR
Permutation and substitution steps repeated 16times
Re-joining of the left and right parts then inverse initialpermutation

Program:

```
package com.islab;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
public class DESEncryptionDecryption
{
 private static Cipher encryptCipher;
 private static Cipher decryptCipher;
        public static void main(String[] args)
        {
                 try
                {
                KeyGenerator keygenerator = KeyGenerator.getInstance("DES");
                SecretKey secretKey = keygenerator.generateKey();
                encryptCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
                encryptCipher.init(Cipher.ENCRYPT_MODE, secretKey);
                byte[] encryptedData = encryptData("Classified Information!");
                decryptCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
                decryptCipher.init(Cipher.DECRYPT_MODE, secretKey);
                decryptData(encryptedData);
```

```
                }
                catch (NoSuchAlgorithmException e)
                {
                 e.printStackTrace();
                 }
                catch (NoSuchPaddingException e)
                {
                e.printStackTrace();
                }
                catch (InvalidKeyException e)
                {
                e.printStackTrace();
                }
                catch (IllegalBlockSizeException e)
                {
                e.printStackTrace();
                }
                catch (BadPaddingException e)
                {
                e.printStackTrace();
                }
        }
 /**
  * Encrypt Data
  * @param data
  * @return
  * @throws IllegalBlockSizeException
  * @throws BadPaddingException
  */
 private static byte[] encryptData(String data) throws IllegalBlockSizeException,
BadPaddingException
{
 System.out.println("Data Before Encryption :" + data);
 byte[] dataToEncrypt = data.getBytes();
  byte[] encryptedData = encryptCipher.doFinal(dataToEncrypt);
  System.out.println("Encryted Data: " + encryptedData);
  return encryptedData;
 }
 /**
  * Decrypt Data
  * @param data
  * @throws IllegalBlockSizeException
  * @throws BadPaddingException
  */
 private static void decryptData(byte[] data) throws IllegalBlockSizeException,
  BadPaddingException
  {
 byte[] textDecrypted = decryptCipher.doFinal(data);
 System.out.println("Decryted Data: " + new String(textDecrypted));
 }}
```

Output:

Data Before Encryption : Classified Information!
Encryted Data: [B@7bea5671
Decryted Data: Classified Information!

TASK 7

Write a JAVA Program to implement the RSA algorithm.

Aim: To implement RSA Algorithm.

Description:

RSA is an algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of them can be given to everyone. A basic principle behind RSA is the observation that it is practical to find three very large positive integers e, d and n such that with modular exponentiation for all integerm:

$$(m^e)^d = m \pmod n$$

The public key is represented by the integers n and e; and, the private key, by the integer d. m represents the message. RSA involves a public key and a private key. The public key can be known by everyone and is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time using the privatekey

Program:

```
package com.islab;
import java.math.BigInteger;
import java.util.Random;
import java.io.*;
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 int blocksize = 256;
 //blocksize in byte
 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;

        }
 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());
 System.out.println("Encrypted String in Bytes: " + bytesToString(encrypted));
 // decrypt
 byte[] decrypted = rsa.decrypt(encrypted);
 System.out.println("Decrypted String in 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();
}
}
```

Output:

```
Enter the plain text:
hello
Encrypting String: hello
String in Bytes: 104101108108111
Encrypted String in Bytes: 0-11572-1311781-48-63124-89-307992-218126-54-2436469-81-16-8730764-65-86-37125-1172242-8411791-47-116-794718-109124-3967-10929-45-422109-935220-11-42-93122108-90-91-8691946567180-24-785499-114-37384211956-3018-79-95-57-36-11-17-782-11123101328126-3410220-99-5611566-1453-7523113929781-95718041-35-8610559949-5747-2611360-425728-2610179-92-34121122-50-10961-86-13-33-99-123121103-109341-120-42-999867-89-5433825271-4326962387-86-13-13-5724-7077-75-4940-9969-127112-56-64-48-1022-6-110116-66-40-41-34101-58-123979989-85-22-109110-2729104-63-40-72-989109-75-97-45-2858119104114-5166-5439125-3716-85-103-679110020-436012167-4049-30-58-77-11113-8462-4324-35-548169125-7199-7186
Decrypted String in Bytes: 104101108108111
Decrypted String: hello
```

TASK 8

Implement the Diffie-Hellman Key Exchange mechanism.

Aim: To implement the Diffie-Hellman Key Exchange mechanism.

Description:

Diffie–Hellman Key Exchange establishes a shared secret between two parties that can be used for secret communication for exchanging data over a public network. It is primarily used as a method of exchanging cryptography keys for use in symmetric encryption algorithms like AES. The algorithm in itself is very simple. The process begins by having the two parties, Alice and Bob. Let's assume that Alice wants to establish a shared secret with Bob.

Program:

```
package com.islab;
import java.io.*;
import java.math.BigInteger;
public class DiffieHellman
{
	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("Key1 calculated :"+k1);
	BigInteger k2=R1.modPow(y,p);
	System.out.println("Key2 calculated :"+k2);
	System.out.println("deffie hellman secret key Encryption has Taken");
   }
}
```

Output:

Enter prime number:

13

Enter primitive root of 13:10

Enter value for x less than 13:

10

R1=3

Enter value for y less than 13:11

R2=4

Key1 calculated :9

Key2 calculated :9

deffie hellman secret key Encryption has Taken

TASK 9

Calculate the message digest of a text using the SHA-1 algorithm in JAVA.

Aim: To calculate the message digest of a text using the SHA-1 algorithm in JAVA.

Description:

In cryptography, SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function. SHA-1 produces a 160-bit hash value known as a message digest. The way this algorithm works is that for a message of size < 264 bits it computes a 160-bit condensed output called a message digest. The SHA-1 algorithm is designed so that it is practically infeasible to find two input messages that hash to the same output message. A hash function such as SHA-1 is used to calculate an alphanumeric string that serves as the cryptographic representation of a file or a piece of data. This is called a digest and can serve as a digital signature. It is supposed to be unique andnon-reversible.

Program:

```
package com.islab;
import java.security.*;
public class SHA1
{
public static void main(String[] a)
{
        try
        {
        MessageDigest md = MessageDigest.getInstance("SHA1");
        System.out.println("Message digest object info: ");
        System.out.println("   Algorithm = " + md.getAlgorithm());
        //System.out.println("   Provider = " + md.getProvider());
        System.out.println("   ToString = " + md.toString());
        String input = "";
        md.update(input.getBytes());
        byte[] output = md.digest();
        System.out.println();
        System.out.println("SHA1(\"" + input + "\") = "+ bytesToHex(output));
        input = "abc";
        md.update(input.getBytes());
        output = md.digest();
        System.out.println();
        System.out.println("SHA1(\"" + input + "\") = "+ bytesToHex(output));
        input = "abcdefghijklmnopqrstuvwxyz";
        md.update(input.getBytes());
        output = md.digest();
        System.out.println();
       System.out.println("SHA1(\"" + input + "\") = "+ bytesToHex(output));
       System.out.println("");
        }
        catch (Exception e)
        {
```

```
        System.out.println("Exception: " + e);
        }
        }
    public static String bytesToHex(byte[] b)
    {
    char hexDigit[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',‘A', 'B', 'C', 'D', 'E', 'F' };
    StringBuffer buf = new StringBuffer();
    for (int j = 0; j < b.length; j++)
    {
    buf.append(hexDigit[(b[j] >> 4) & 0x0f]);
    buf.append(hexDigit[b[j] & 0x0f]);
    }
    return buf.toString();
}
}
```

Output:

Message digest object info:
Algorithm=SHA1
Provider=SUN version 1.6
ToString=SHA1 Message Digest from SUN, <initialized>

SHA1("")=DA39A3EE5E6B4B0D3255BFEF95601890AFD80709

SHA1("abc")=A9993E364706816ABA3E25717850C26C9CD0D89D

SHA1("abcdefghijklmnopqrstuvwxyz")=32D10C7B8CF96570CA04CE37F2A1
9D84240D3A89

TASK 10

Calculate the message digest of a text using the MD5 algorithm in JAVA.

Aim: To calculate the message digest of a text using the MD5 algorithm in JAVA.

Description:

MD5 processes a variable-length message into a fixed-length output of 128 bits. The input message is broken up into chunks of 512-bit blocks. The message is padded so that its length is divisible by 512. The padding works as follows: first a single bit, 1, is appended to the end of the message. This is followed by as many zeros as are required to bring the length of the message up to 64bits less than a multiple of 512.The remaining bits are filled up with 64 bits representing the length of the original message, modulo 2^{64}.The main MD5 Algorithm operates on a 128-bit state, divided into four 32-bit words, denoted A, B, C, and D. These are initialized to certain fixed constants. The main algorithm then uses each 512-bit message block in turn to modify the state .

M

Program:

```
package com.islab;
import java.security.*;
public class MD5
{
public static void main(String[] a)
{
        try
        {
                MessageDigest md = MessageDigest.getInstance("MD5");
                System.out.println("Message digest object info: ");
                System.out.println("   Algorithm = " + md.getAlgorithm());
                // System.out.println("   Provider = " +md.getProvider());
                System.out.println("   ToString = " + md.toString());
                String input = "";
                md.update(input.getBytes());
                byte[] output = md.digest();
                System.out.println();
                System.out.println("MD5(\"" + input + "\") = " + bytesToHex(output));
                input = "abc";
                md.update(input.getBytes());
                output = md.digest();
                System.out.println();
                System.out.println("MD5(\"" + input + "\") = " + bytesToHex(output));
                input = "abcdefghijklmnopqrstuvwxyz";
                md.update(input.getBytes());
                output = md.digest();
                System.out.println();
                System.out.println("MD5(\"" + input + "\") = " + bytesToHex(output));
                System.out.println("");
        }
        catch (Exception e)
        {
                System.out.println("Exception: " + e);
```

```
                }
        }
public static String bytesToHex(byte[] b)
{
char hexDigit[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9','A', 'B', 'C', 'D', 'E', 'F' };
StringBuffer buf = new StringBuffer();
for (int j = 0; j < b.length; j++)
{
buf.append(hexDigit[(b[j] >> 4) & 0x0f]);
buf.append(hexDigit[b[j] & 0x0f]);
}
return buf.toString();
}
}
```

Output:

Message digest object info:
  Algorithm = MD5
  ToString = MD5 Message Digest from SUN, <initialized>

MD5("") = D41D8CD98F00B204E9800998ECF8427E

MD5("abc") = 900150983CD24FB0D6963F7D28E17F72

MD5("abcdefghijklmnopqrstuvwxyz") =
C3FCD3D76192E4007DFB496CCA67E13B

TASK 11

Explore the java classes related to Digital Certificates.

Aim: To explore the Java classes related to Digital Certificates.

The use and verification of digital signatures is another standard engine that is included in the security provider architecture. Like the other engines we've examined, the classes that implement this engine have both a public interface and an SPI for implementors of the engine.
In the JDK, the most common use of digital signatures is to create signed classes; users have the option of granting additional privileges to these signed classes using the mechanics of the access controller. In addition, a security manager and a class loader can use this information to change the policy of the security manager.

The Signature Class

Operations on digital signatures are abstracted by the `Signature` class (`java.security.Signature`):

public abstract class Signature extends SignatureSpi

Provide an engine to create and verify digital signatures. In Java 1.1, there is no `SignatureSpi` class, and this class simply extends the `Object` class.
The Sun security provider includes a single implementation of this class that generates signatures based on the DSA algorithm.

Using the Signature Class

As with all engine classes, instances of the `Signature` class are obtained by calling one of these methods:

public static Signature getInstance(String algorithm)
public static Signature getInstance(String algorithm, String provider)

Generate a signature object that implements the given algorithm. If no provider is specified, all providers are searched in order for the given algorithm otherwise, the system searches for the given algorithm only in the given provider. If an implementation of the given algorithm is not found, a `NoSuchAlgorithmException` is thrown. If the named security provider cannot be found, a `NoSuchProviderException` is thrown.
If the algorithm string is "DSA", the string "SHA/DSA" is substituted for it. Hence, implementors of this class that provide support for DSA signing must register themselves appropriately (that is, with the message digest algorithm name) in the security provider.
Once a signature object is obtained, the following methods can be invoked on it:

public void final initVerify(PublicKey publicKey)

Initialize the signature object, preparing it to verify a signature. A signature object must be initialized before it can be used. If the key is not of the correct type for the algorithm or is otherwise invalid, an `InvalidKeyException` is thrown.

public final void initSign(PrivateKey privateKey)

Initialize the signature object, preparing it to create a signature. A signature object must be initialized before it can be used. If the key is not of the correct type for the algorithm or is otherwise invalid, an `InvalidKeyException` is thrown.

public final void update(byte b)
public final void update(byte[] b)
public final void update(byte b[], int offset, int length)

Add the given data to the accumulated data the object will eventually sign or verify. If the object has not been initialized, a `SignatureException` is thrown.

public final byte[] sign()
public final int sign(byte[] outbuf, int offset, int len) ★
Create the digital signature, assuming that the object has been initialized for signing. If the object has not been properly initialized, a `SignatureException` is thrown. Once the signature has been generated, the object is reset so that it may generate another signature based on some new data (however, it is still initialized for signing; a new call to the `initSign()` method is not required).
In the first of these methods, the signature is returned from the method. Otherwise, the signature is stored into the `outbuf` array at the given offset, and the length of the signature is returned. If the output buffer is too small to hold the data, an `IllegalArgumentException` will be thrown.
public final boolean verify(byte[] signature)
Test the validity of the given signature, assuming that the object has been initialized for verification. If the object has not been properly initialized, then a `SignatureException` is thrown. Once the signature has been verified (whether or not the verification succeeds), the object is reset so that it may verify another signature based on some new data (no new call to the `initVerify()` method is required).
public final String getAlgorithm()
Get the name of the algorithm this object implements.
public String toString()
A printable version of a signature object is composed of the string "`Signature object:`" followed by the name of the algorithm implemented by the object, followed by the initialized state of the object. The state is either `<not initialized>`, `<initialized for verifying>`, or `<initialized for signing>`. However, the Sun DSA implementation of this class overrides this method to show the parameters of the DSA algorithm instead.
public final void setParameter(String param, Object value) ☆
public final void setParameter(AlgorithmParameterSpec param) ★
Set the parameter of the signature engine. In the first format, the named parameter is set to the given value; in the second format, parameters are set based on the information in the `param` specification.
In the Sun implementation of the DSA signing algorithm, the only valid `param` string is `KSEED`, which requires an array of bytes that will be used to seed the random number generator used to generate the `k` value. There is no way to set this value through the parameter specification, which in the Sun implementation always returns an `UnsupportedOperationException`.
public final Object getParameter(String param) ☆
Return the named parameter from the object. The only valid string for the Sun implementation is `KSEED`.
public final Provider getProvider() ★
Return the provider that supplied the implementation of this signature object.
It is no accident that this class has many similarities to the `MessageDigest` class; a digital signature algorithm is typically implemented by performing a cryptographic operation on a private key and the message digest that represents the data to be signed. For the developer, this means that generating a digital signature is virtually the same as generating a message digest; the only difference is that a key must be presented in order to operate on a signature object. This difference is important, however, since it fills in the hole we noticed previously: a message digest can be altered along with the data it represents so that the tampering is

unnoticeable. A signed message digest, on the other hand, can't be altered without knowledge of the key that was used to create it. The use of a public key in the digital signature algorithm makes the digital signature more attractive than a message authentication code, in which there must be a shared key between the parties involved in the message exchange.

Program:

Class Definition

```
public class Send
{
public static void main(String args[])
{
String data;
data = "This have I thought good to deliver thee, " +"that thou mightst not lose the dues of
rejoicing " +"by being ignorant of what greatness is promised thee.";
            try
            {
            FileOutputStream fos = new FileOutputStream("test");
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());
            ks.load(new FileInputStream(System.getProperty("user.home") +File.separator
    +".keystore"), null);
            char c[] = new char[args[1].length()];
            args[1].getChars(0, c.length, c, 0);
            PrivateKey pk = (PrivateKey) ks.getKey(args[0], c);
            Signature s = Signature.getInstance("DSA");
            s.initSign(pk);
            byte buf[] = data.getBytes();
            s.update(buf);
            oos.writeObject(data);
            oos.writeObject(s.sign());
            }
            catch (Exception e)
            {
            e.printStackTrace();
            }
}
}
```

The SignedObject Class

In our last example, we had to create an object that held both the data in which we are interested and the signature for that data. This is a common enough requirement that Java provides the `SignedObject` class (`java.security.SignedObject`) to encapsulate an object and its signature:

public final class SignedObject implements Serializable ★

Encapsulate an object and its digital signature. The encapsulated object must be serializable so that a serialization of a signed object can do a deep copy of the embedded object.
Signed objects are created with this constructor:

public SignedObject(Serializable o, PrivateKey pk, Signature engine) ★
Create a signed object based on the given object, signing the serialized data in that object with the given private key and signature object. The signed object contains a copy of the given object; this copy is obtained by serializing the object parameter. If this serialization fails, an `IOException` is thrown.
It's very important to realize that this constructor makes, in effect, a copy of its parameter; if you create a signed object based on a string buffer and later change the contents of the string buffer, the data in the signed object remains unchanged. This preserves the integrity of the object encapsulated with its signature.
Here are the methods we can use to operate on a signed object:
public Object getContent() ★
Return the object embedded in the signed object. The object is reconstituted using object serialization; an error in serialization may cause either an `IOException` or a `ClassNotFoundException` to be thrown.
public byte[] getSignature() ★
Return the signature embedded in the signed object.
public String getAlgorithm() ★
Return the name of the algorithm that was used to sign the object.
public boolean verify(PublicKey pk, Signature s) ★
Verify the signature within the embedded object with the given key and signature engine. The signature engine parameter may be obtained by calling the `getInstance()` method of the `Signature` class. The underlying signature engine may throw an `InvalidKeyException` or `SignatureException`.

TASK 12

Write a program in java, which performs a digital signature on a given text.

Aim: To write a program which performs a digital signature on a given text.

Program:

```
package com.islab;
import java.security.*;
public class Digital{

   public static void main(String[] args) {
      try {
         // Generating key pair
         KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
         keyPairGenerator.initialize(2048);
         KeyPair keyPair = keyPairGenerator.generateKeyPair();

         // Creating a Signature object
         Signature signature = Signature.getInstance("SHA256withRSA");

         // Initializing the signature
         signature.initSign(keyPair.getPrivate());

         // Data to be signed
         String data = "Hello, this is the text to be signed.";

         // Update the data to be signed
         signature.update(data.getBytes());

         // Generate the digital signature
         byte[] digitalSignature = signature.sign();

         // Print the digital signature
         System.out.println("Digital Signature: " + new String(digitalSignature));

         // Verification
         Signature verificationSignature = Signature.getInstance("SHA256withRSA");
         verificationSignature.initVerify(keyPair.getPublic());
         verificationSignature.update(data.getBytes());

         // Verify the digital signature
         boolean isVerified = verificationSignature.verify(digitalSignature);
         System.out.println("Signature verified: " + isVerified);
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

Output:

Digital Signature:

`
`
`
`
`
`
`

Signature verified: true