EXPERIMENT - 1

AIM: Write a C program that contains a string (char pointer) with a value 'Hello world'. The program should XOR each character in this string with 0 and displays the result.

DESCRIPTION:

An XOR gate is a digital logic gate with two or more inputs and one output that performs exclusive disjunction.

ALGORITHM:

1. Start
2. Take input as "Hello World" which is assigned to variable "str"
3. Initialise the variable 'len' for calculating the length of the string
4. Print the length of the word followed by the input

PROGRAM:

```
#include<stdlib.h>
int main(){
        char str[]="Hello World";
        char str1[11];
        int i,len;
        len=strlen(str);
        for(i=0;i<len;i++) {
                str1[i]=str[i]^0;
                printf("%c",str1[i]);
        }
        printf("\n");
}
```

OUTPUT:

Hello World
Hello World

EXPERIMENT - 2

AIM: Write a C program that contains a string (char pointer) with a value 'Hello world'. The program should AND or and XOR each character in this string with 127 and display the result.

DESCRIPTION:

The AND gate is an electronic circuit that gives a high output(1) only if all the inputs are high.A dot(.) is used to show the AND operation i.e. A.B this dot is sometimes omitted.

ALGORITHM:

1. Start
2. Take the input 'hello world' which is assigned to variable 'str'
3. Perform AND operation between the string and 127.
4. Then print the result
5. Stop.

PROGRAM:

```
#include <stdio.h>
#include<stdlib.h>
void main(){
        char str[]="Hello World";
        char str1[11];
        char str2[11]=str[];
         int i,len;
         len = strlen(str);
        for(i=0;i<len;i++) {
                str1[i] = str[i]&127;
                printf("%c",str1[i]);
        }
        printf("\n");
```

```
        for(i=0;i<len;i++) {
                str3[i] = str2[i]^127;
                printf("%c",str3[i]);
        }
        printf("\n");
}
```

Output:

Hello World

Hello World

Hello World

EXPERIMENT – 3

AIM: Write a Java program to perform encryption and decryption using the following algorithms

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

a) Ceaser Cipher

DESCRIPTION:

The Caesar cipher technique is one of the earliest and simplest method of encryption technique .It's simple type of substitution cipher i.e. each letter of given text is replaced by a letter, some fixed number of positions down the alphabet. Thus, to cipher a given text, we need an integer value, known as shift which indicates the number of position each letter of the text has been moved down. The encryption can be replaced by using modular arithmetic by first transforming the letter into numbers according to the scheme A=0,B=1,…. Y=24,Z=25.

e.g. ABCD; shift=4 Cipher: EFGH

ALGORITHM:

1. Start
2. Read the string
3. Traverse the given text one character at a time
4. For each character transforms the given character as per the key
5. It prints the encryption and decryption of the given string
6. Stop

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 World

Enter the Key: 5

Encrypted String is: MjqqtBtwqi

Decrypted String is: Hello World

b) Substitution Cipher

DESCRIPTION:

Substitution Cipher is a method of encrypting by which units of plain text are replaced with cipher text,the ‘units’ may be single letter or mixture of the letters.In a substitution cipher,the units of the plain text are retained in the same sequence in the cipher text,but the units themselves are altered.

ALGORITHM:

1. Start
2. Assign a= ‘abcdefghijklmnopqrstuvwxyz’ b= ‘zyxwvutsrqponmkjihgfedcba’
3. Read input string which need to be encrypted
4. Compare variables ‘a’ and ‘b’ values and substitute the input string position with variable ‘b’strings till end of input string
5. Print cipher text which is produced from step-4
6. Stop

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 {
        // TODO code application logic here
                String a = "abcdefghijklmnopqrstuvwxyz";
                String b = "zyxwvutsrqponmlkjihgfedcba";
                System.out.print("Enter any string: ");
                String str = br.readLine();
                String decrypt = "";
                char c;
                for(int i=0;i<str.length();i++) {
```

```
                c = str.charAt(i);
                int j = a.indexOf(c);
                decrypt = decrypt+b.charAt(j);
            }
            System.out.println("The encrypted data is: " +decrypt);
        }
}
```

OUTPUT:

Enter any string: aceho

The encrypted data is: zxvsl

c) Hill Cipher

DESCRIPTION:

Hill cipher is a polygraph substitution cipher based on linear algebra.Each letter is represented by a number modulo 26.To encrypt a message,each block of 'n'letters is multiplied by an inversible n*n matrix against modulus 26.To decrypt the message,each block is multiplied by the inverse of the matrix used for encryption.The matrix used for encryption is the cipher key and it should be chosen randomly from the set of inversible n*n.

ALGORITHM:

1. Start
2. To encrypt a string each block of '3'letters are multiplied by inversible 3*3matrix against %26
3. To decrypt a string each block is multiplied by inverse of the matrix used for encryption
4. Read a three-letter string
5. It pints the inverse matrix, encryption and decryption of the given three letter string
6. Stop.

PROGRAM:

```
import java.io.*;
import java.util.*;
import java.io.*;
public class HillCipher {
static float[][] decrypt = new float[3][1];
static float[][] a = new float[3][3];
static float[][] b = new float[3][3];
static float[][] mes = new float[3][1];
static float[][] res = new float[3][1];
static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        static Scanner sc = new Scanner(System.in);
        public static void main(String[] args) throws IOException {
        // TODO code application logic here
```

```
getkeymes();
for(int i=0;i<3;i++)
        for(int j=0;j<1;j++)
                for(int k=0;k<3;k++) {
                        res[i][j]=res[i][j]+a[i][k]*mes[k][j];
                }
                System.out.print("\nEncrypted string is : ");
                for(int i=0;i<3;i++) {
                        System.out.print((char)(res[i][0]%26+97));
                        res[i][0]=res[i][0];
                }
                inverse();
                for(int i=0;i<3;i++)
                        for(int j=0;j<1;j++)
                                for(int k=0;k<3;k++) {
                                        decrypt[i][j] = decrypt[i][j]+b[i][k]*res[k][j];
                                }
                                System.out.print("\nDecrypted string is : ");
                                for(int i=0;i<3;i++){
                                        System.out.print((char)(decrypt[i][0]%26+97));
                                }
                                System.out.print("\n");
        }
        public static void getkeymes() throws IOException {
                System.out.println("Enter 3x3 matrix for key (It should be inversible): ");
                for(int i=0;i<3;i++)
                        for(int j=0;j<3;j++)
                                a[i][j] = sc.nextFloat();
                                System.out.print("\nEnter a 3 letter string: ");
                                String msg = br.readLine();
                                for(int i=0;i<3;i++)
```

```
                    mes[i][0] = msg.charAt(i)-97;
}
public static void inverse() {
        floatp,q;
        float[][] c = a;
        for(int i=0;i<3;i++)
                for(int j=0;j<3;j++) {
                        //a[i][j]=sc.nextFloat();
                        if(i==j)
                                b[i][j]=1;
                        else
                                b[i][j]=0;
                }
        for(int k=0;k<3;k++) {
                for(int i=0;i<3;i++) {
                        p = c[i][k];
                        q = c[k][k];
                        for(int j=0;j<3;j++) {
                                if(i!=k) {
                                        c[i][j] = c[i][j]*q-p*c[k][j];
                                        b[i][j] = b[i][j]*q-p*b[k][j];
                        }
                 }
         }
}
for(int i=0;i<3;i++)
        for(int j=0;j<3;j++) {
                b[i][j] = b[i][j]/c[i][i];
        }
System.out.println("");
System.out.println("\nInverse Matrix is : ");
```

```
                for(int i=0;i<3;i++) {
                        for(int j=0;j<3;j++)
                        System.out.print(b[i][j] + " ");
                        System.out.print("\n");
                }
        }
}
```

OUTPUT:

Enter a 3 letter string: hai

Encrypted string is :fdx

Inverse Matrix is :

0.083333336 0.41666666 -0.33333334

-0.41666666 -0.083333336 0.6666667

0.5833333 -0.083333336 -0.33333334

Decrypted string is :hai

EXPERIMENT – 4

AIM: Write a C/JAVA program to implement the DES algorithm logic.

DESCRIPTION:

DES is an implementation of Feistel Cipher. It uses16 round Feistel structure. The block size is 64 bits. Though, key length is 64-bits, DES has an effective key length and 56-bits. Since 8 of the 64bits of the key one not used by the encryption algorithm.

ALGORITHM:

1. First we need to get the key generator instance using DES algorithm.
2. Generate secure key that will be used for encryption and decryption.
3. Get Cipher instance using DES algorithm. One for encrypt mode and another for decrypt mode. Initialize the cipher object using key and IVParameterSpec object.
4. For encryption, create object of CipherOutputStream using encrypt cipher. For decryption, create object of CipherInputStream using decrypt cipher.
5. Read the input stream and write the output stream.
6. Stop

PROGRAM:

```
import java.util.*;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.security.spec.KeySpec;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.DESedeKeySpec;
import sun.misc.BASE64Decoder;
import sun.misc.BASE64Encoder;
public class DES {
```

```
        private static final String UNICODE_FORMAT = "UTF8";
        public static final String DESEDE_ENCRYPTION_SCHEME = "DESede";
        privateKeySpecmyKeySpec;
        privateSecretKeyFactorymySecretKeyFactory;
        private Cipher cipher;
        byte[] keyAsBytes;
        private String myEncryptionKey;
        private String myEncryptionScheme;
        SecretKey key;
        static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        public DES() throws Exception {
                // TODO code application logic here
                myEncryptionKey = "ThisIsSecretEncryptionKey";
                myEncryptionScheme = DESEDE_ENCRYPTION_SCHEME;
                keyAsBytes = myEncryptionKey.getBytes(UNICODE_FORMAT);
                myKeySpec = new DESedeKeySpec(keyAsBytes);
                mySecretKeyFactory = SecretKeyFactory.getInstance(myEncryptionScheme);
                cipher = Cipher.getInstance(myEncryptionScheme);
                key = mySecretKeyFactory.generateSecret(myKeySpec);
}
        public String encrypt(String unencryptedString) {
                String encryptedString = null;
                try {
                        cipher.init(Cipher.ENCRYPT_MODE, key);
                        byte[] plainText = unencryptedString.getBytes(UNICODE_FORMAT);
                        byte[] encryptedText = cipher.doFinal(plainText);
                        BASE64Encoder base64encoder = new BASE64Encoder();
                        encryptedString = base64encoder.encode(encryptedText);
                }
                catch (Exception e) {
                        e.printStackTrace();
```

```
        }
        returnencryptedString;
    }
    public String decrypt(String encryptedString) {
        String decryptedText=null;
        try {
            cipher.init(Cipher.DECRYPT_MODE, key);
            BASE64Decoder base64decoder = new BASE64Decoder();
            byte[] encryptedText = base64decoder.decodeBuffer(encryptedString);
            byte[] plainText = cipher.doFinal(encryptedText);
            decryptedText= bytes2String(plainText); }
        catch (Exception e) {
            e.printStackTrace();
        }
        returndecryptedText;
    }
    private static String bytes2String(byte[] bytes) {
        StringBufferstringBuffer = new StringBuffer();
        for (int i = 0; i <bytes.length; i++) {
            stringBuffer.append((char) bytes[i]);
        }
        returnstringBuffer.toString();
    }
    public static void main(String args []) throws Exception {
        System.out.print("Enter the string: ");
        DES myEncryptor= new DES();
        String stringToEncrypt = br.readLine();
        String encrypted = myEncryptor.encrypt(stringToEncrypt);
        String decrypted = myEncryptor.decrypt(encrypted);
        System.out.println("\nString To Encrypt: " +stringToEncrypt);
        System.out.println("\nEncrypted Value : " +encrypted);
```

```
            System.out.println("\nDecrypted Value : " +decrypted);
            System.out.println("");
        }
}
```

OUTPUT:

```
Enter the string: Welcome
String To Encrypt: Welcome
Encrypted Value : BPQMwc0wKvg=
Decrypted Value : Welcome
```

EXPERIMENT – 5

AIM: Write a C/JAVA program to implement the Blowfish algorithm logic.

DESCRIPTION:

Blowfish is a symmetric key block cipher. Blowfish provides a good encryption rate in software and no effective cryptanalysis of it has been found to date.

Blowfish has a 64bit block size and a variable key length from 32bits up to 448bits. It is a 16-round Feistel cipher and uses large key-dependent s-boxes.

ALGORITHM:

1. Blowfish has a 64-bit block size and a variable key length from 30bits up to 48bits.
2. It is a 16-round Feistel cipher and uses large key dependent s-boxes.
3. There are 5 sub key-arrays. One 18-entry p-array and four 256-entry s-boxes. 4.Every round r consists of 4 actions.

a) XOR the left half of the data with the 'r'th p-array entry.

b) Use the XORed data as input for Blowfish algorithm. c)F-function's output with the right half (R) of the data. d)Swap L and R.

5. The F-function splits the 32bits into four 8-bits quarters and uses the quarters as input to the s-boxes.
6. The s-boxes accept 8-bit input and produce 32-bit output.The outputs are added modulo 2 power 32 and XORed to produce the final 32-bit output.

PROGRAM:

```
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
		KeyGeneratorkeyGenerator = 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("inputFile.txt");
		FileOutputStreamfout = new FileOutputStream("outputFile.txt");
		CipherOutputStreamcout = 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: dI1MXzW97oQ=

Contents of inputFile.txt: Hello World

Contents of outputFile.txt: ùJÖ˜ NåI“

EXPERIMENT - 6

AIM: Write a C/JAVA program to implement the Rijndael algorithm logic.

DESCRIPTION:

The more popular and widely adopted symmetric encryption algorithm likely to be encountered now-a-days in the Advanced Encryption Standard(AES). It is performed at least 6 times faster than the triple DES.

ALGORITHM:

1. Derive the set of round keys from the cipher key
2. Initialize the state array with the block data(plain text)
3. Add the initial round key to the starting state array 4.perform nine rounds of state manipulation
5. Perform the tenth and final round of state manipulation
6. Copy the final state array out as the encrypted data(cipher text)
7. Stop

PROGRAM:

```
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import java.io.*;
public class AES {
        public static String asHex (byte buf[]) {
                StringBuffer strbuf = new StringBuffer(buf.length * 2);
                int i;
                for (i = 0; i < buf.length; i++) {
                        if (((int) buf[i] & 0xff) < 0x10)
                                strbuf.append("0");
                        strbuf.append(Long.toString((int) buf[i] & 0xff, 16));
                }
```

```
            return strbuf.toString();
        }
        public static void main(String[] args) throws Exception {
            String message="AES still rocks!!";
            // Get the KeyGenerator
            KeyGenerator kgen = KeyGenerator.getInstance("AES");
            kgen.init(128); // 192 and 256 bits may not be available
            // Generate the secret key specs.
            SecretKey skey = kgen.generateKey();
            byte[] raw = skey.getEncoded();
            SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
            // Instantiate the cipher
            Cipher cipher = Cipher.getInstance("AES");
            cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
            byte[] encrypted = cipher.doFinal((args.length == 0 ? message
:args[0]).getBytes());
            System.out.println("encrypted string: " + asHex(encrypted));
            cipher.init(Cipher.DECRYPT_MODE, skeySpec);
            byte[] original = cipher.doFinal(encrypted);
            String originalString=new String(original);
            System.out.println("Original string: " + originalString + " " + asHex(original));
        }
}
```

OUTPUT:

```
Enter the string: Welcome
String To Encrypt: Welcome
Encrypted Value : BPQGHKIoMNMwc0wKvg=
```

EXPERIMENT – 7

AIM: Write the RC4 logic in Java Using Java cryptography; encrypt the text "Hello world" using Blowfish. Create your own key using Java key tool.

DESCRIPTION:

RC4 is a stream cipher, symmetric key algorithm. The same algorithm can be used for encryption and decryption as the data stream is simply XORed with the generated key sequence. The key stream is completely independent of the plain text used. It uses a variable length key from 1 to 256 bit to initialize a 256-bit state table. The state table is used for subsequent generation of pseudo-random bits and then to generate a pseudo-random stream which is XORed with the plain text to give the cipher text.

ALGORITHM:

1. Get the data to be encrypted and the selected key 2.create two string arrays
3. Initiate one array with numbers from 0 to 255
4. Fill the other array with the selected key
5. Randomize the first array depending on the array of the key
6. Randomize the first array within itself to generate the final key stream
7. XOR the final key stream with the data to be encrypted to give the cipher text.

PROGRAM:

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.swing.JOptionPane;
public class BlowFishCipher {
        public static void main(String[] args) throws Exception {
                // create a key generator based upon the Blowfish cipher
               KeyGeneratorkeygenerator = KeyGenerator.getInstance("Blowfish");
                // create a key
```

```
                SecretKeysecretkey = keygenerator.generateKey();
                // create a cipher based upon Blowfish
                Cipher cipher = Cipher.getInstance("Blowfish");
                // initialise cipher to with secret key
                cipher.init(Cipher.ENCRYPT_MODE, secretkey);
                // get the text to encrypt
                String inputText = JOptionPane.showInputDialog("Input your message: ");
                // encrypt message
                byte[] encrypted = cipher.doFinal(inputText.getBytes());
                // re-initialise the cipher to be in decrypt mode
                cipher.init(Cipher.DECRYPT_MODE, secretkey);
                // decrypt message
                byte[] decrypted = cipher.doFinal(encrypted);
                 // and display the results
                JOptionPane.showMessageDialog(JOptionPane.getRootFrame(),"\nEncrypted
text: " + new String(encrypted) + "\n" +  "\nDecrypted text: " + new String(decrypted));
                System.exit(0);
        }
}
```

OUTPUT:

Input your message: Hello world

Encrypted text: 3ooo&&(*&*4r4

Decrypted text: Hello world

EXPERIMENT - 8

AIM: Implement the Diffie-Hellman Key Exchange mechanism using HTML and JavaScript.

DESCRIPTION:

Diffie-Hellman key exchange(DH) is a method of securely exchanging cryptographic keys over a public channel and was one of the first public-key protocols as originally conceptualized by Ralph Merkle and named after Whitfield Diffie and Martin Hellman.

ALGORITHM:

1. Select prime number i.e. q
2. Find the primitive root of q i.e. x
3. Assume private key for user 'A' is x. Calculating public key for user A is YA=αXAmodq
4. Assuming private key is XB. Calculating public key is YB, YB=αXBmodq 5. Generating secret key K, K=(YA)XBmodq, K=(YB)XAmodq
6. Stop

PROGRAM:

```
import java.math.BigInteger;
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.SecureRandom;
import javax.crypto.spec.DHParameterSpec;
import javax.crypto.spec.DHPublicKeySpec;
public class DiffeHellman {
        public final static int pValue = 47;
        public final static int gValue = 71;
        public final static int XaValue = 9;
        public final static int XbValue = 14;
```

```
    public static void main(String[] args) throws Exception {
        // TODO code application logic here
        BigInteger p = new BigInteger(Integer.toString(pValue));
            BigInteger g = new BigInteger(Integer.toString(gValue));
            BigIntegerXa = new BigInteger(Integer.toString(XaValue));
            BigIntegerXb = new BigInteger(Integer.toString(XbValue));
            createKey();
            intbitLength = 512; // 512 bits
            SecureRandomrnd = new SecureRandom();
            p = BigInteger.probablePrime(bitLength, rnd);
            g = BigInteger.probablePrime(bitLength, rnd);
            createSpecificKey(p, g);
    }
    public static void createKey() throws Exception {
        KeyPairGeneratorkpg = KeyPairGenerator.getInstance("DiffieHellman");
        kpg.initialize(512);
        KeyPairkp = kpg.generateKeyPair();
        KeyFactorykfactory = KeyFactory.getInstance("DiffieHellman");
    DHPublicKeySpeckspec=(DHPublicKeySpec)kfactory.getKeySpec(kp.getPublic(),
DHPublicKeySpec.class);
        System.out.println("Public key is: " +kspec);
    }
    public static void createSpecificKey(BigInteger p, BigInteger g) throws Exception {
        KeyPairGeneratorkpg = KeyPairGenerator.getInstance("DiffieHellman");
        DHParameterSpecparam = new DHParameterSpec(p, g);
        kpg.initialize(param);
        KeyPairkp = kpg.generateKeyPair();
        KeyFactorykfactory = KeyFactory.getInstance("DiffieHellman");
            DHPublicKeySpeckspec = (DHPublicKeySpec)
          kfactory.getKeySpec(kp.getPublic(),DHPublicKeySpec.class);
            System.out.println("\nPublic key is : " +kspec);
```

```
        }
}
```

OUTPUT:

```
Public key is: javax.crypto.spec.DHPublicKeySpec@5afd29
Public key is: javax.crypto.spec.DHPublicKeySpec@9971ad
```

EXPERIMENT - 9

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

DESCRIPTION:

In cryptography, SHA-1(Secure Hashing Algorithm) is a cryptographic hash function which takes an input and produces a 160-bit(20-byte) hash value known as message digest typically rendered as a hexadecimal number, 40 digits long.

ALGORITHM:

1. Pad the bit 00… so that length of plain text is 128<Multiple of 1024 bits
2. Append 128 bits representing of original text such that length=Multiple of 1024 bits
3. Initiate the buffers(a, b, c, d, e, f, g and h). Each buffer size=64 bits in hexadecimal
4. Process each block of plain text in 80 rounds 5.output buffers is Hash code which has length of 1 bit.

PROGRAM:

```
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'};
            StringBufferbuf = new StringBuffer();
            for (int j=0; j<b.length; j++) {
                buf.append(hexDigit[(b[j] >> 4) & 0x0f]);
                buf.append(hexDigit[b[j] & 0x0f]);
            }
            returnbuf.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")=32D10C7B8CF96570CA04CE37F2A19D8424
0D3A89

EXPERIMENT - 10

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

DESCRIPTION:

The MD5 message-digest algorithm is widely a used hash function producing a 128-bit hash value. Although MD5 was initially designed to be as a cryptographic hash function, it has been found to suffer from extensive vulnerabilities. It can still be used as a checksum to verify data integrity, but only against unintentional corruption. It remains suitable for other non-cryptographic purposes, for example for determining the partition for a particular key in a partitioned database.

ALGORITHM:

1. The input message is broken up into chunks of 512-bit blocks(sixteen 32-bit words), the message is padded so that its length is divisible by 512
2. First a single bit,1 is appended to the end of message and followed by as many as zeros required to bring the length of the message upto 64 bits fewer than a multiple of 512
3. Remaining bits are filled up with 64 bits representing length of original message, modulo 264
4. MD5 operates on 128-bit state, divided into four 32-bit words, denoted A, B, C and D
5. Algorithm then uses 512-bit message block to modify the state

PROGRAM:

```
import java.security.*;
public class MD5 {
        public static void main(String[] a) {
                 // TODO code application logic here
                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'};
        StringBufferbuf = 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

Provider = SUN version 1.6

ToString = MD5 Message Digest from SUN, <initialized>

MD5("") = D41D8CD98F00B204E9800998ECF8427E

MD5("abc") = 900150983CD24FB0D6963F7D28E17F72

MD5("abcdefghijklmnopqrstuvwxyz") = C3FCD3D76192E4007DFB496CCA67E13B

DESCRIPTION: Substitution Cipher is a method of encrypting by which units of plain text are replaced with cipher text,the ‘units’ may be single letter or mixture of the letters.In a substitution cipher,the units of the plain text are retained in the same sequence in the cipher text,but the units themselves are altered

```
import java.util.Scanner;

public class SubstitutionCipher {

    // Define the substitution mappings
    private static final String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    private static final String SUBSTITUTION =
"QWERTYUIOPASDFGHJKLZXCVBNM";

    public static String encrypt(String plaintext) {
        plaintext = plaintext.toUpperCase(); // Convert to uppercase
        StringBuilder ciphertext = new StringBuilder();

        for (char c : plaintext.toCharArray()) {
            if (ALPHABET.indexOf(c) != -1) {
                // Substitute the character
                int index = ALPHABET.indexOf(c);
                ciphertext.append(SUBSTITUTION.charAt(index));
            } else {
                // Keep non-alphabetic characters unchanged
                ciphertext.append(c);
            }
```

```
    }
    return ciphertext.toString();
}

public static String decrypt(String ciphertext) {
    ciphertext = ciphertext.toUpperCase(); // Convert to uppercase
    StringBuilder plaintext = new StringBuilder();

    for (char c : ciphertext.toCharArray()) {
        if (SUBSTITUTION.indexOf(c) != -1) {
            // Reverse the substitution
            int index = SUBSTITUTION.indexOf(c);
            plaintext.append(ALPHABET.charAt(index));
        } else {
            // Keep non-alphabetic characters unchanged
            plaintext.append(c);
        }
    }
    return plaintext.toString();
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.println("Enter the text to encrypt:");
    String plaintext = scanner.nextLine();
```

```
        String encryptedText = encrypt(plaintext);
        System.out.println("Encrypted text: " + encryptedText);

        String decryptedText = decrypt(encryptedText);
        System.out.println("Decrypted text: " + decryptedText);

        scanner.close();
    }
}
```

7. Write a Java program to implement RSA algorithm.

```
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.Scanner;

public class RSA {
    private BigInteger n; // Modulus
    private BigInteger e; // Public exponent
    private BigInteger d; // Private exponent
    private int bitLength = 1024; // Length of key in bits

    // Constructor to generate keys
    public RSA() {
        SecureRandom random = new SecureRandom();
        BigInteger p = BigInteger.probablePrime(bitLength / 2, random);
```

```
        BigInteger q = BigInteger.probablePrime(bitLength / 2, random);
        n = p.multiply(q);
        BigInteger phi =
(p.subtract(BigInteger.ONE)).multiply(q.subtract(BigInteger.ONE));

        // Choose public exponent e
        e = BigInteger.probablePrime(bitLength / 2, random);
        while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) < 0)
{
            e = e.add(BigInteger.ONE);
        }

        // Compute private key d
        d = e.modInverse(phi);
    }

    // Encryption
    public BigInteger encrypt(BigInteger message) {
        return message.modPow(e, n);
    }

    // Decryption
    public BigInteger decrypt(BigInteger ciphertext) {
        return ciphertext.modPow(d, n);
    }
    public BigInteger getN() {
        return n;
```

```
    }

    public BigInteger getE() {
        return e;
    }

    // Main method
    public static void main(String[] args) {
        RSA rsa = new RSA();
        Scanner scanner = new Scanner(System.in);

        System.out.println("RSA Key Generation:");
        System.out.println("Public Key (n, e): (" + rsa.getN() + ", " + rsa.getE() +")");
        System.out.println("Private Key (d): [hidden for security]");
        System.out.print("Enter a message (as an integer): ");
        BigInteger message = scanner.nextBigInteger();

        // Encryption
        BigInteger ciphertext = rsa.encrypt(message);
        System.out.println("Encrypted Message: " + ciphertext);
        BigInteger decryptedMessage = rsa.decrypt(ciphertext);
        System.out.println("Decrypted Message: " + decryptedMessage);

        scanner.close();
    }
}
```