

L D COLLEGE OF ENGINEERING AHMEDABAD
COMPUTER ENGINEERING DEPARTMENT

VISION:

To achieve academic excellence in Computer Engineering by providing value based education.

MISSION:

1. To produce graduates according to the needs of industry, government, society and scientific community.
2. To develop partnership with industries, research and development organizations and government sectors for continuous improvement of faculties and students.
3. To motivate students for participating in reputed conferences, workshops, seminars and technical events to make them technocrats and entrepreneurs.
4. To enhance the ability of students to address the real life issues by applying technical expertise, human values and professional ethics.
5. To inculcate habit of using free and open source software, latest technology and soft skills so that they become competent professionals.
6. To encourage faculty members to upgrade their skills and qualification through training and higher studies at reputed universities.

LABORATORY PLANNING

Dept: Computer Engineering SEMESTER: BE-VII

TERM: Jun-2020

SUBJECT: -- Information and Network Security (2170709) Div: A & B

Faculty: Prof. Rajyalakshmi Jaiswal

Sr. No.	Title	Planned date
1	Implement Caesar cipher encryption-decryption.	Week 3 Jun 18
2	Implement Mono alphabetic cipher encryption-decryption.	Week 4 Jun 18
3	Implement Playfair cipher encryption-decryption.	Week 1 Jul 18
4	Implement Poly alphabetic cipher encryption-decryption.	Week 2 Jul 18
5	Implement Hill cipher encryption-decryption.	Week 3 Jul 18
6	To implement Simple DES or AES.	Week 1 Aug 18
7	Implement Diffi-Hellmen Key exchange Method.	Week 2 Aug 18
8	Implement RSA encryption-decryption algorithm.	Week 3 Aug 18
9	Write a program to generate SHA-1 hash.	Week 1 Sep 18
10	Implement a digital signature algorithm.	Week 2 Sep 18
11	Perform various encryption-decryption techniques with cryptool.	Week 3 Sep 18
12	Study and use the Wire shark for the various network protocols.	Week 1 Oct 18

INDEX

Sr.N o.	Title	Page No.	Date	MARKS OUT OF				Grade	Sign
				20					
				C1 (5)	C2 (5)	C3 (5)	C4 (5)		
1	Implement Caesar cipher encryption-decryption.		31/08/2020						
2	Implement Mono alphabetic cipher encryption-decryption.		31/08/2020						
3	Implement Playfair cipher encryption-decryption.		31/08/2020						
4	Implement Poly alphabetic cipher encryption-decryption.		31/08/2020						
5	Implement Hill cipher encryption-decryption.		31/08/2020						
6	To implement Simple DES or AES.		31/08/2020						
7	Implement Diffi-Hellmen Key exchange Method.		31/08/2020						
8	Implement RSA encryption-decryption algorithm.		10/10/2020						
9	Write a program to generate SHA-1 hash.		10/10/2020						
10	Implement a digital signature algorithm.		10/10/2020						
11	Perform various encryption-decryption techniques with cryptool.		10/10/2020						
12	Study and use the Wire shark for the various network protocols.		10/10/2020						

Grading Criteria (C1) Understanding Concepts (C2) Logic Designing (C3) Program Execution (C4) Final Output	Grading Scale : A – Above 17 (Excellent) B – 13 to 16 (Good) C – 10 to 12 (Fair) D – Below 10 (Poor) A, B, C are Passing Grades. D is Failing Grade.
---	--

Practical 1

Aim: Implement Caesar cipher encryption-decryption.

Objective: The action of a Caesar cipher is to replace each plaintext letter with a different one a fixed number of places down the alphabet. The cipher illustrated here uses a left shift of three, so that (for example) each occurrence of E in the plaintext becomes B in the ciphertext.

Theory:

The Caesar Cipher technique is one of the earliest and simplest method of encryption technique. It's simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter some fixed number of positions down the alphabet. For example with a shift of 1, A would be replaced by B, B would become C, and so on. The method is apparently named after Julius Caesar, who apparently used it to communicate with his officials.

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 represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1,..., Z = 25. Encryption of a letter by a shift n can be described mathematically as.

$$E_n(x)=(x+n)\bmod 26$$

(Encryption Phase with shift n)

$$D_n(x)=(x-n)\bmod 26$$

(Decryption Phase with shift n)

Program :

```
import java.util.*;

public class Main
{
    public static final String ALPHABET = "abcdefghijklmnopqrstuvwxyz";

    public static String encrypt(String plainText, int shiftKey)
    {
        plainText = plainText.toLowerCase();
        String cipherText = "";
        for (int i = 0; i < plainText.length(); i++)
        {
            int charPosition = ALPHABET.indexOf(plainText.charAt(i));
            int keyVal = (shiftKey + charPosition) % 26;
            char replaceVal = ALPHABET.charAt(keyVal);
            cipherText += replaceVal;
        }
        return cipherText;
    }

    public static String decrypt(String cipherText, int shiftKey)
    {
        cipherText = cipherText.toLowerCase();
        String plainText = "";
        for (int i = 0; i < cipherText.length(); i++)
        {
            int charPosition = ALPHABET.indexOf(cipherText.charAt(i));
            int keyVal = (charPosition - shiftKey) % 26;
            if (keyVal < 0)
```

```
{
    keyVal = ALPHABET.length() + keyVal;
}
char replaceVal = ALPHABET.charAt(keyVal);
plainText += replaceVal;
}
return plainText;
}
public static void main(String[] args)
{
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the String for Encryption: ");
    String message = new String();
    message = sc.next();
    System.out.println("Cipher Text: " + encrypt(message, 3));
    System.out.println("Plain Text: " + decrypt(encrypt(message, 3), 3));
    sc.close();
}
}
```

Output :

```
Enter the String for Encryption:
sdbasmbdkjasd
Cipher Text: vegedvpqegnmdvg
Plain Text: sdbasmbdkjasd

...Program finished with exit code 0
Press ENTER to exit console.□
```


Practical 2

Aim: Implement Mono alphabetic cipher encryption-decryption.

Objective: For each character of alphabet, assign different-abrupt concerned character. Monoalphabetic ciphers are easy to break because they reflect the frequency data of the original alphabet. A countermeasure is to provide multiple substitutes, known as homophones, for a single letter.

Theory:

Substitution ciphers are probably the most common form of cipher. They work by replacing each letter of the plaintext (and sometimes punctuation marks and spaces) with another letter (or possibly even a random symbol).

A *monoalphabetic substitution cipher*, also known as a simple substitution cipher, relies on a fixed replacement structure. That is, the substitution is fixed for each letter of the alphabet. Thus, if "a" is encrypted to "R", then every time we see the letter "a" in the plaintext, we replace it with the letter "R" in the ciphertext.

A simple example is where each letter is encrypted as the next letter in the alphabet: "a simple message" becomes "B TJNQMF NFTTBHF". In general, when performing a simple substitution manually, it is easiest to generate the *ciphertext alphabet* first, and encrypt by comparing this to the plaintext alphabet. The table below shows how one might choose to, and we will, lay them out for this example.

Plaintext Alphabet	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
Ciphertext Alphabet	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	A

Program :

```
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.util.HashSet;
import java.util.LinkedHashSet;
import java.util.Set;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Main {

    private static final String alphabet = "abcdefghijklmnopqrstuvwxyz";

    public String getCharsPermutation() {

        Set set = new LinkedHashSet();
        SecureRandom random;
        StringBuilder chars = new StringBuilder();
        try {
            random = SecureRandom.getInstance("SHA1PRNG");
            while (set.size() < 26) {
                set.add(alphabet.charAt(random.nextInt(alphabet.length())));
            }
            for (Object object : set) {
                chars.append(object);
            }
        } catch (NoSuchAlgorithmException ex) {
```

```
        System.out.println("No such Algorithm");
    }
    return chars.toString();
}

public String encrypt(String plaintext, String key) {
    StringBuilder ciphertext = new StringBuilder();

    for (char chr : plaintext.toLowerCase().toCharArray()) {
        byte position = (byte) alphabet.indexOf(chr);
        if (chr == ' ') {
            ciphertext.append(" ");
        } else {
            ciphertext.append(key.charAt(position));
        }
    }
    return ciphertext.toString();
}

public String decrypt(String ciphertext, String key) {
    StringBuilder plaintext = new StringBuilder();
    for (char chr : ciphertext.toLowerCase().toCharArray()) {
        byte position = (byte) key.indexOf(chr);
        if (chr == ' ') {
            plaintext.append(" ");
        } else {
            plaintext.append(alphabet.charAt(position));
        }
    }
}
```

```
    }  
    return plaintext.toString();  
}  
  
public static void main(String[] args) {  
    String plaintext="monoalphabetic encryption algorithm";  
    Main mono = new Main();  
    String key=mono.getCharsPermutation();  
    System.out.println("Plaintext : "+plaintext +"\nAlphabet : "+alphabet+"\nkey      :  
"+key);  
    String ciphertext=mono.encrypt(plaintext, key);  
    System.out.println("Ciphertext message: "+ciphertext);  
    System.out.println("Recovered message : " +mono.decrypt(ciphertext,key ));  
}  
}
```

Output :

```
Plaintext : monoalphabetic encryption algorithm  
Alphabet   : abcdefghijklmnopqrstuvwxyz  
key        : mzungkctywhxosilafnepbvjdq  
Ciphertext message: oisimxltmzgeyu gsufdleyis mxcifyeto  
Recovered message : monoalphabetic encryption algorithm  
  
...Program finished with exit code 0  
Press ENTER to exit console.□
```

Practical 3

Aim: Implement Playfair cipher encryption-decryption.

Objective: The technique encrypts pairs of letters (bigrams or digrams), instead of single letters as in the simple substitution cipher and rather more complex Vigenère cipher systems then in use. The Playfair is thus significantly harder to break since the frequency analysis used for simple substitution ciphers does not work with it.

Theory:

The Playfair cipher uses a 5 by 5 table containing a key word or phrase. Memorization of the keyword and 4 simple rules was all that was required to create the 5 by 5 table and use the cipher.

The Algorithm consists of 2 steps:

1. Generate the key Square(5×5):

The key square is a 5×5 grid of alphabets that acts as the key for encrypting the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table (as the table can hold only 25 alphabets). If the plaintext contains J, then it is replaced by I.

The initial alphabets in the key square are the unique alphabets of the key in the order in which they appear followed by the remaining letters of the alphabet in order.

2. Algorithm to encrypt the plain text:

The plaintext is split into pairs of two letters (digraphs). If there is an odd number of letters, a Z is added to the last letter.

Program :

```
import java.util.Scanner;

public class Main
{
    private String KeyWord  = new String();
    private String Key  = new String();
    private char  matrix_arr[][] = new char[5][5];
    public void setKey(String k)
    {
        String K_adjust = new String();
        boolean flag = false;
        K_adjust = K_adjust + k.charAt(0);
        for (int i = 1; i < k.length(); i++)
        {
            for (int j = 0; j < K_adjust.length(); j++)
            {
                if (k.charAt(i) == K_adjust.charAt(j))
                {
                    flag = true;
                }
            }
            if (flag == false)
                K_adjust = K_adjust + k.charAt(i);
            flag = false;
        }
        KeyWord = K_adjust;
    }
}
```

```
}  
public void KeyGen()  
{  
    boolean flag = true;  
    char current;  
    Key = KeyWord;  
    for (int i = 0; i < 26; i++)  
    {  
        current = (char) (i + 97);  
        if (current == 'j')  
            continue;  
        for (int j = 0; j < KeyWord.length(); j++)  
        {  
            if (current == KeyWord.charAt(j))  
            {  
                flag = false;  
                break;  
            }  
        }  
        if (flag)  
            Key = Key + current;  
        flag = true;  
    }  
    System.out.println(Key);  
    matrix();  
}  
private void matrix()
```

```
{
    int counter = 0;
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            matrix_arr[i][j] = Key.charAt(counter);
            System.out.print(matrix_arr[i][j] + " ");
            counter++;
        }
        System.out.println();
    }
}
```

```
private String format(String old_text)
{
    int i = 0;
    int len = 0;
    String text = new String();
    len = old_text.length();
    for (int tmp = 0; tmp < len; tmp++)
    {
        if (old_text.charAt(tmp) == 'j')
        {
            text = text + 'i';
        }
        else
```



```
        text = text + old_text.charAt(tmp);
    }
    len = text.length();
    for (i = 0; i < len; i = i + 2)
    {
        if (text.charAt(i + 1) == text.charAt(i))
        {
            text = text.substring(0, i + 1) + 'x' + text.substring(i + 1);
        }
    }
    return text;
}
```

```
private String[] Divid2Pairs(String new_string)
{
    String Original = format(new_string);
    int size = Original.length();
    if (size % 2 != 0)
    {
        size++;
        Original = Original + 'x';
    }
    String x[] = new String[size / 2];
    int counter = 0;
    for (int i = 0; i < size / 2; i++)
    {
        x[i] = Original.substring(counter, counter + 2);
```

```
        counter = counter + 2;
    }
    return x;
}

public int[] GetDiminsions(char letter)
{
    int[] key = new int[2];
    if (letter == 'j')
        letter = 'i';
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            if (matrix_arr[i][j] == letter)
            {
                key[0] = i;
                key[1] = j;
                break;
            }
        }
    }
    return key;
}

public String encryptMessage(String Source)
{
    String src_arr[] = Divid2Pairs(Source);
    String Code = new String();
```

```
char one;
char two;
int part1[] = new int[2];
int part2[] = new int[2];
for (int i = 0; i < src_arr.length; i++)
{
    one = src_arr[i].charAt(0);
    two = src_arr[i].charAt(1);
    part1 = GetDiminsions(one);
    part2 = GetDiminsions(two);
    if (part1[0] == part2[0])
    {
        if (part1[1] < 4)
            part1[1]++;
        else
            part1[1] = 0;
        if (part2[1] < 4)
            part2[1]++;
        else
            part2[1] = 0;
    }
    else if (part1[1] == part2[1])
    {
        if (part1[0] < 4)
            part1[0]++;
        else
            part1[0] = 0;
```

```
        if (part2[0] < 4)
            part2[0]++;
        else
            part2[0] = 0;
    }
    else
    {
        int temp = part1[1];
        part1[1] = part2[1];
        part2[1] = temp;
    }
    Code = Code + matrix_arr[part1[0]][part1[1]]
        + matrix_arr[part2[0]][part2[1]];
}
return Code;
}
public String decryptMessage(String Code)
{
    String Original = new String();
    String src_arr[] = Divid2Pairs(Code);
    char one;
    char two;
    int part1[] = new int[2];
    int part2[] = new int[2];
    for (int i = 0; i < src_arr.length; i++)
    {
        one = src_arr[i].charAt(0);
```

```
two = src_arr[i].charAt(1);
part1 = GetDiminsions(one);
part2 = GetDiminsions(two);
if (part1[0] == part2[0])
{
    if (part1[1] > 0)
        part1[1]--;
    else
        part1[1] = 4;
    if (part2[1] > 0)
        part2[1]--;
    else
        part2[1] = 4;
}
else if (part1[1] == part2[1])
{
    if (part1[0] > 0)
        part1[0]--;
    else
        part1[0] = 4;
    if (part2[0] > 0)
        part2[0]--;
    else
        part2[0] = 4;
}
else
{
```

```
        int temp = part1[1];
        part1[1] = part2[1];
        part2[1] = temp;
    }
    Original = Original + matrix_arr[part1[0]][part1[1]]
        + matrix_arr[part2[0]][part2[1]];
    }
    return Original;
}

public static void main(String[] args)
{
    Main x = new Main();
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter a keyword:");
    String keyword = sc.next();
    x.setKey(keyword);
    x.KeyGen();
    System.out
        .println("Enter word to encrypt: (Make sure length of message is even)");
    String key_input = sc.next();
    if (key_input.length() % 2 == 0)
    {
        System.out.println("Encryption: " + x.encryptMessage(key_input));
        System.out.println("Decryption: "
            + x.decryptMessage(x.encryptMessage(key_input)));
    }
    else
```

```
System.out.println("Message length should be even");
```

```
sc.close();
```

```
}
```

```
}
```

Output :

```
Enter a keyword:
SendMe
SendMabcfghiklmopqrstuvwxyz
S e n d M
a b c f g
h i k l m
o p q r s
t u v w x
Enter word to encrypt: (Make sure length of message is even)
learningcenter
Encryption: idfoekMcbnSvdp
Decryption: learningcenter

...Program finished with exit code 0
Press ENTER to exit console.□
```

Practical 4

Aim: Implement Poly alphabetic cipher encryption-decryption

Objective: Polyalphabetic ciphers have the advantage over simple substitution ciphers that symbol frequencies are not preserved. Under different alphabets, the same plaintext character is thus encrypted to different ciphertext characters, precluding simple frequency analysis as per mono-alphabetic substitution.

Theory:

Vigenère Cipher

Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of polyalphabetic substitution. A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets. The encryption of the original text is done using the Vigenère square or Vigenère table.

The table consists of the alphabets written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar Ciphers.

Encryption

The first letter of the plaintext, G is paired with A, the first letter of the key. So use row G and column A of the Vigenère square, namely G. Similarly, for the second letter of the plaintext, the second letter of the key is used, the letter at row E and column Y is C. The rest of the plaintext is enciphered in a similar fashion.

Decryption

Decryption is performed by going to the row in the table corresponding to the key, finding the position of the ciphertext letter in this row, and then using the column's label as the plaintext. For example, in row A (from AYUSH), the ciphertext G appears in column G, which is the first plaintext letter. Next we go to row Y (from AYUSH), locate the ciphertext C which is found in column E, thus E is the second plaintext letter.

Program :

```
public class Main
{
    public static String encrypt(String text, final String key)
    {
        String res = "";
        text = text.toUpperCase();
        for (int i = 0, j = 0; i < text.length(); i++)
        {
            char c = text.charAt(i);
            if (c < 'A' || c > 'Z')
                continue;
            res += (char) ((c + key.charAt(j) - 2 * 'A') % 26 + 'A');
            j = ++j % key.length();
        }
        return res;
    }

    public static String decrypt(String text, final String key)
    {
        String res = "";
        text = text.toUpperCase();
        for (int i = 0, j = 0; i < text.length(); i++)
        {
            char c = text.charAt(i);
            if (c < 'A' || c > 'Z')
                continue;
```

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

public static void main(String[] args)
{
    String key = "VIGENERECIPHER";
    String message = "Beware the Jabberwock, my son!";
    String encryptedMsg = encrypt(message, key);
    System.out.println("String: " + message);
    System.out.println("Encrypted message: " + encryptedMsg);
    System.out.println("Decrypted message: " + decrypt(encryptedMsg, key));
}
}
```

Output :

```
String: Beware the Jabberwock, my son!
Encrypted message: WMCEEIKLGRPIFVMEUGXQPWQV
Decrypted message: BEWARETHEJABBERWOCKMYSON

...Program finished with exit code 0
Press ENTER to exit console. □
```

Practical 5

Aim: Implement Hill cipher encryption-decryption.

Objective: Hill Cipher is the application of modulo arithmetic to cryptography. This cryptographic technique uses a square matrix as the key used to encrypt and decrypt [5]. Security is expected to be guaranteed after applying the Hill Cipher algorithm in the process of sending and receiving data.

Theory:

Hill cipher is a polygraphic substitution cipher based on linear algebra. Each letter is represented by a number modulo 26. Often the simple scheme A = 0, B = 1, ..., Z = 25 is used, but this is not an essential feature of the cipher. To encrypt a message, each block of n letters (considered as an n -component vector) is multiplied by an invertible $n \times n$ 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 invertible $n \times n$ matrices (modulo 26).

Program :

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Main
{
    int keymatrix[][];
    int linematrix[];
    int resultmatrix[];

    public void divide(String temp, int s)
    {
        while (temp.length() > s)
        {
            String sub = temp.substring(0, s);
            temp = temp.substring(s, temp.length());
            perform(sub);
        }
        if (temp.length() == s)
            perform(temp);
        else if (temp.length() < s)
        {
            for (int i = temp.length(); i < s; i++)
                temp = temp + 'x';
            perform(temp);
        }
    }
}
```

```
    }  
}
```

```
public void perform(String line)  
{  
    linetomatrix(line);  
    linemultiplykey(line.length());  
    result(line.length());  
}
```

```
public void keytomatrix(String key, int len)  
{  
    keymatrix = new int[len][len];  
    int c = 0;  
    for (int i = 0; i < len; i++)  
    {  
        for (int j = 0; j < len; j++)  
        {  
            keymatrix[i][j] = ((int) key.charAt(c)) - 97;  
            c++;  
        }  
    }  
}
```

```
public void linetomatrix(String line)  
{  
    linematrix = new int[line.length()];
```

```
    for (int i = 0; i < line.length(); i++)
    {
        linematrix[i] = ((int) line.charAt(i)) - 97;
    }
}

public void linemultiplykey(int len)
{
    resultmatrix = new int[len];
    for (int i = 0; i < len; i++)
    {
        for (int j = 0; j < len; j++)
        {
            resultmatrix[i] += keymatrix[i][j] * linematrix[j];
        }
        resultmatrix[i] %= 26;
    }
}

public void result(int len)
{
    String result = "";
    for (int i = 0; i < len; i++)
    {
        result += (char) (resultmatrix[i] + 97);
    }
    System.out.print(result);
}
```

```
}
```

```
public boolean check(String key, int len)
```

```
{
```

```
    keytomatrix(key, len);
```

```
    int d = determinant(keymatrix, len);
```

```
    d = d % 26;
```

```
    if (d == 0)
```

```
    {
```

```
        System.out
```

```
            .println("Invalid key!!! Key is not invertible because determinant=0...");
```

```
        return false;
```

```
    }
```

```
    else if (d % 2 == 0 || d % 13 == 0)
```

```
    {
```

```
        System.out
```

```
            .println("Invalid key!!! Key is not invertible because determinant has  
common factor with 26...");
```

```
        return false;
```

```
    }
```

```
    else
```

```
    {
```

```
        return true;
```

```
    }
```

```
}
```

```
public int determinant(int A[], int N)
```

```
{
    int res;
    if (N == 1)
        res = A[0][0];
    else if (N == 2)
    {
        res = A[0][0] * A[1][1] - A[1][0] * A[0][1];
    }
    else
    {
        res = 0;
        for (int j1 = 0; j1 < N; j1++)
        {
            int m[][] = new int[N - 1][N - 1];
            for (int i = 1; i < N; i++)
            {
                int j2 = 0;
                for (int j = 0; j < N; j++)
                {
                    if (j == j1)
                        continue;
                    m[i - 1][j2] = A[i][j];
                    j2++;
                }
            }
            res += Math.pow(-1.0, 1.0 + j1 + 1.0) * A[0][j1]
                * determinant(m, N - 1);
        }
    }
}
```



```
    }  
    }  
    return res;  
}
```

```
public void cofact(int num[], int f)
```

```
{  
    int b[], fac[];  
    b = new int[f][f];  
    fac = new int[f][f];  
    int p, q, m, n, i, j;  
    for (q = 0; q < f; q++)  
    {  
        for (p = 0; p < f; p++)  
        {  
            m = 0;  
            n = 0;  
            for (i = 0; i < f; i++)  
            {  
                for (j = 0; j < f; j++)  
                {  
                    b[i][j] = 0;  
                    if (i != q && j != p)  
                    {  
                        b[m][n] = num[i][j];  
                        if (n < (f - 2))  
                            n++;  
                    }  
                }  
            }  
        }  
    }  
}
```

```
        else
        {
            n = 0;
            m++;
        }
    }
}

fac[q][p] = (int) Math.pow(-1, q + p) * determinant(b, f - 1);
}
}
trans(fac, f);
}
```

```
void trans(int fac[], int r)
{
    int i, j;
    int b[], inv[];
    b = new int[r][r];
    inv = new int[r][r];
    int d = determinant(keymatrix, r);
    int mi = mi(d % 26);
    mi %= 26;
    if (mi < 0)
        mi += 26;
    for (i = 0; i < r; i++)
    {
```

```
        for (j = 0; j < r; j++)
        {
            b[i][j] = fac[j][i];
        }
    }
    for (i = 0; i < r; i++)
    {
        for (j = 0; j < r; j++)
        {
            inv[i][j] = b[i][j] % 26;
            if (inv[i][j] < 0)
                inv[i][j] += 26;
            inv[i][j] *= mi;
            inv[i][j] %= 26;
        }
    }
    System.out.print("\nInverse key: ");
    matrixtoinvkey(inv, r);
}

public int mi(int d)
{
    int q, r1, r2, r, t1, t2, t;
    r1 = 26;
    r2 = d;
    t1 = 0;
    t2 = 1;
```

```
while (r1 != 1 && r2 != 0)
{
    q = r1 / r2;
    r = r1 % r2;
    t = t1 - (t2 * q);
    r1 = r2;
    r2 = r;
    t1 = t2;
    t2 = t;
}
return (t1 + t2);
}
```

```
public void matrixtoinvkey(int inv[], int n)
{
    String invkey = "";
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            invkey += (char) (inv[i][j] + 97);
        }
    }
    System.out.print(invkey);
}
```

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

```
{
    Main obj = new Main();
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    int choice;
    System.out.println("Menu:\n1: Encryption\n2: Decryption");
    choice = Integer.parseInt(in.readLine());
    System.out.println("Enter the line: ");
    String line = in.readLine();
    System.out.println("Enter the key: ");
    String key = in.readLine();
    double sq = Math.sqrt(key.length());
    if (sq != (long) sq)
        System.out
            .println("Invalid key length!!! Does not form a square matrix...");
    else
    {
        int s = (int) sq;
        if (obj.check(key, s))
        {
            System.out.print("Result: ");
            obj.divide(line, s);
            obj.cofact(obj.keymatrix, s);
        }
    }
}
```

Output :

```
Menu:
1: Encryption
2: Decryption
1
Enter the line:
shortexample
Enter the key:
hill
Result: apadjtftwlfj
Inverse key: zwbx

...Program finished with exit code 0
Press ENTER to exit console.█
```

```
Menu:
1: Encryption
2: Decryption
2
Enter the line:
apadjtftwlfj
Enter the key:
zwbx
Result: shortexample
Inverse key: hill

...Program finished with exit code 0
Press ENTER to exit console.█
```

Practical 6

Aim: To implement Simple DES or AES.

Data Encryption Standard (DES) :

Objective: DES is a block cipher algorithm that takes plain text in blocks of 64 bits and converts them to ciphertext using keys of 48 bits. It is a symmetric key algorithm, which means that the same key is used for encrypting and decrypting data. Encryption and decryption using the DES algorithm.

Theory:

Data encryption standard (DES) has been found vulnerable against very powerful attacks and therefore, the popularity of DES has been found slightly on decline.

DES is a block cipher, and encrypts data in blocks of size of 64 bit each, means 64 bits of plain text goes as the input to DES, which produces 64 bits of cipher text. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bits.

The Broad-level steps in DES :

1. In the first step, the 64 bit plain text block is handed over to an initial Permutation (IP) function.
2. The initial permutation performed on plain text.
3. Next the initial permutation (IP) produces two halves of the permuted block; says Left Plain Text (LPT) and Right Plain Text (RPT).
4. Now each LPT and RPT to go through 16 rounds of encryption process.
5. In the end, LPT and RPT are rejoined and a Final Permutation (FP) is performed on the combined block
6. The result of this process produces 64 bit cipher text.

Program :

```
import java.util.*;
```

```
class Main {
```

```
    private static class DES {
```

```
        // Initial Permutation Table
```

```
        int[] IP = { 58, 50, 42, 34, 26, 18,  
                    10, 2, 60, 52, 44, 36, 28, 20,  
                    12, 4, 62, 54, 46, 38,  
                    30, 22, 14, 6, 64, 56,  
                    48, 40, 32, 24, 16, 8,  
                    57, 49, 41, 33, 25, 17,  
                    9, 1, 59, 51, 43, 35, 27,  
                    19, 11, 3, 61, 53, 45,  
                    37, 29, 21, 13, 5, 63, 55,  
                    47, 39, 31, 23, 15, 7 };
```

```
        // Inverse Initial Permutation Table
```

```
        int[] IP1 = { 40, 8, 48, 16, 56, 24, 64,  
                    32, 39, 7, 47, 15, 55,  
                    23, 63, 31, 38, 6, 46,  
                    14, 54, 22, 62, 30, 37,  
                    5, 45, 13, 53, 21, 61,  
                    29, 36, 4, 44, 12, 52,  
                    20, 60, 28, 35, 3, 43,  
                    11, 51, 19, 59, 27, 34,
```



```
2, 42, 10, 50, 18, 58,  
26, 33, 1, 41, 9, 49,  
17, 57, 25 };
```

```
// first key-hePermutation Table
```

```
int[] PC1 = { 57, 49, 41, 33, 25,  
17, 9, 1, 58, 50, 42, 34, 26,  
18, 10, 2, 59, 51, 43, 35, 27,  
19, 11, 3, 60, 52, 44, 36, 63,  
55, 47, 39, 31, 23, 15, 7, 62,  
54, 46, 38, 30, 22, 14, 6, 61,  
53, 45, 37, 29, 21, 13, 5, 28,  
20, 12, 4 };
```

```
// second key-Permutation Table
```

```
int[] PC2 = { 14, 17, 11, 24, 1, 5, 3,  
28, 15, 6, 21, 10, 23, 19, 12,  
4, 26, 8, 16, 7, 27, 20, 13, 2,  
41, 52, 31, 37, 47, 55, 30, 40,  
51, 45, 33, 48, 44, 49, 39, 56,  
34, 53, 46, 42, 50, 36, 29, 32 };
```

```
// Expansion D-box Table
```

```
int[] EP = { 32, 1, 2, 3, 4, 5, 4,  
5, 6, 7, 8, 9, 8, 9, 10,  
11, 12, 13, 12, 13, 14, 15,  
16, 17, 16, 17, 18, 19, 20,
```

```
21, 20, 21, 22, 23, 24, 25,  
24, 25, 26, 27, 28, 29, 28,  
29, 30, 31, 32, 1 };
```

```
// Straight Permutation Table
```

```
int[] P = { 16, 7, 20, 21, 29, 12, 28,  
            17, 1, 15, 23, 26, 5, 18,  
            31, 10, 2, 8, 24, 14, 32,  
            27, 3, 9, 19, 13, 30, 6,  
            22, 11, 4, 25 };
```

```
// S-box Table
```

```
int[][][] sbox = {  
    { { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7 },  
      { 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8 },  
      { 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0 },  
      { 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 } },  
  
    { { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10 },  
      { 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5 },  
      { 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15 },  
      { 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 } },  
  
    { { 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8 },  
      { 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1 },  
      { 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7 },  
      { 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 } },  
  
    { { 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15 },
```

```

        { 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9 },
        { 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4 },
        { 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14 } },
    { { 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9 },
      { 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6 },
      { 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14 },
      { 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 } },
    { { 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11 },
      { 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8 },
      { 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6 },
      { 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13 } },
    { { 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1 },
      { 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6 },
      { 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2 },
      { 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12 } },
    { { 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7 },
      { 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2 },
      { 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8 },
      { 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 } }
};

int[] shiftBits = { 1, 1, 2, 2, 2, 2, 2, 2,
                    1, 2, 2, 2, 2, 2, 2, 1 };

// hexadecimal to binary conversion
String hextoBin(String input)
{
    int n = input.length() * 4;

```

```
    input = Long.toBinaryString(
        Long.parseUnsignedLong(input, 16));
    while (input.length() < n)
        input = "0" + input;
    return input;
}

// binary to hexadecimal conversion
String binToHex(String input)
{
    int n = (int)input.length() / 4;
    input = Long.toHexString(
        Long.parseUnsignedLong(input, 2));
    while (input.length() < n)
        input = "0" + input;
    return input;
}

// per-mutate input hexadecimal
// according to specified sequence
String permutation(int[] sequence, String input)
{
    String output = "";
    input = hextoBin(input);
    for (int i = 0; i < sequence.length; i++)
        output += input.charAt(sequence[i] - 1);
    output = binToHex(output);
}
```

```
        return output;
    }

    // xor 2 hexadecimal strings
    String xor(String a, String b)
    {
        // hexadecimal to decimal(base 10)
        long t_a = Long.parseUnsignedLong(a, 16);
        // hexadecimal to decimal(base 10)
        long t_b = Long.parseUnsignedLong(b, 16);
        // xor
        t_a = t_a ^ t_b;
        // decimal to hexadecimal
        a = Long.toHexString(t_a);
        // prepend 0's to maintain length
        while (a.length() < b.length())
            a = "0" + a;
        return a;
    }

    // left Circular Shifting bits
    String leftCircularShift(String input, int numBits)
    {
        int n = input.length() * 4;
        int perm[] = new int[n];
        for (int i = 0; i < n - 1; i++)
            perm[i] = (i + 2);
    }
}
```

```
    perm[n - 1] = 1;
    while (numBits-- > 0)
        input = permutation(perm, input);
    return input;
}

// preparing 16 keys for 16 rounds
String[] getKeys(String key)
{
    String keys[] = new String[16];
    // first key permutation
    key = permutation(PC1, key);
    for (int i = 0; i < 16; i++) {
        key = leftCircularShift(
            key.substring(0, 7), shiftBits[i])
            + leftCircularShift(key.substring(7, 14),
                shiftBits[i]);

        // second key permutation
        keys[i] = permutation(PC2, key);
    }
    return keys;
}

// s-box lookup
String sBox(String input)
{
    String output = "";
```

```
input = hextoBin(input);
for (int i = 0; i < 48; i += 6) {
    String temp = input.substring(i, i + 6);
    int num = i / 6;
    int row = Integer.parseInt(
        temp.charAt(0) + "" + temp.charAt(5), 2);
    int col = Integer.parseInt(
        temp.substring(1, 5), 2);
    output += Integer.toHexString(
        sbox[num][row][col]);
}
return output;
}
```

```
String round(String input, String key, int num)
{
    // fk
    String left = input.substring(0, 8);
    String temp = input.substring(8, 16);
    String right = temp;
    // Expansion permutation
    temp = permutation(EP, temp);
    // xor temp and round key
    temp = xor(temp, key);
    // lookup in s-box table
    temp = sBox(temp);
    // Straight D-box
```

```
temp = permutation(P, temp);
// xor
left = xor(left, temp);
System.out.println("Round "
    + (num + 1) + " "
    + right.toUpperCase()
    + " " + left.toUpperCase() + " "
    + key.toUpperCase());

// swapper
return right + left;
}

String encrypt(String plainText, String key)
{
    int i;
    // get round keys
    String keys[] = getKeys(key);

    // initial permutation
    plainText = permutation(IP, plainText);
    System.out.println(
        "After initial permutation: "
        + plainText.toUpperCase());
    System.out.println(
        "After splitting: L0="
        + plainText.substring(0, 8).toUpperCase())
```



```
        + " R0="
        + plainText.substring(8, 16).toUpperCase() + "\n");

// 16 rounds
for (i = 0; i < 16; i++) {
    plainText = round(plainText, keys[i], i);
}

// 32-bit swap
plainText = plainText.substring(8, 16)
            + plainText.substring(0, 8);

// final permutation
plainText = permutation(IP1, plainText);
return plainText;
}

String decrypt(String plainText, String key)
{
    int i;

    // get round keys
    String keys[] = getKeys(key);

    // initial permutation
    plainText = permutation(IP, plainText);
    System.out.println(
        "After initial permutation: "
```

```
        + plainText.toUpperCase());
    System.out.println(
        "After splitting: L0="
        + plainText.substring(0, 8).toUpperCase()
        + " R0=" + plainText.substring(8, 16).toUpperCase()
        + "\n");

    // 16-rounds
    for (i = 15; i > -1; i--) {
        plainText = round(plainText, keys[i], 15 - i);
    }

    // 32-bit swap
    plainText = plainText.substring(8, 16)
        + plainText.substring(0, 8);
    plainText = permutation(IP1, plainText);
    return plainText;
}
}

public static void main(String args[])
{
    String text = "123456ABCD132536";
    String key = "AABB09182736CCDD";

    System.out.println(
        "Plain Text: " + text.toUpperCase());
    System.out.println(
```

```
        "Key Text: " + text.toUpperCase() + "\n");

DES cipher = new DES();
System.out.println("Encryption:\n");
text = cipher.encrypt(text, key);
System.out.println(
    "\nCipher Text: " + text.toUpperCase() + "\n");
System.out.println("Decryption\n");
text = cipher.decrypt(text, key);
System.out.println(
    "\nPlain Text: "
    + text.toUpperCase());
    }
}
```

Output :

Plain Text: 123456ABCD132536

Key Text: 123456ABCD132536

Encryption:

After initial permutation: 14A7D67818CA18AD

After splitting: L0=14A7D678 R0=18CA18AD

Round 1 18CA18AD 5A78E394 194CD072DE8C
Round 2 5A78E394 4A1210F6 4568581ABCCE
Round 3 4A1210F6 B8089591 06EDA4ACF5B5
Round 4 B8089591 236779C2 DA2D032B6EE3
Round 5 236779C2 A15A4B87 69A629FEC913
Round 6 A15A4B87 2E8F9C65 C1948E87475E
Round 7 2E8F9C65 A9FC20A3 708AD2DDB3C0
Round 8 A9FC20A3 308BEE97 34F822F0C66D
Round 9 308BEE97 10AF9D37 84BB4473DCCC
Round 10 10AF9D37 6CA6CB20 02765708B5BF
Round 11 6CA6CB20 FF3C485F 6D5560AF7CA5
Round 12 FF3C485F 22A5963B C2C1E96A4BF3
Round 13 22A5963B 387CCDAA 99C31397C91F
Round 14 387CCDAA BD2DD2AB 251B8BC717D0
Round 15 BD2DD2AB CF26B472 3330C5D9A36D
Round 16 CF26B472 19BA9212 181C5D75C66D

Cipher Text: C0B7A8D05F3A829C

Decryption

After initial permutation: 19BA9212CF26B472

After splitting: L0=19BA9212 R0=CF26B472

Round 1 CF26B472 BD2DD2AB 181C5D75C66D
Round 2 BD2DD2AB 387CCDAA 3330C5D9A36D
Round 3 387CCDAA 22A5963B 251B8BC717D0
Round 4 22A5963B FF3C485F 99C31397C91F
Round 5 FF3C485F 6CA6CB20 C2C1E96A4BF3
Round 6 6CA6CB20 10AF9D37 6D5560AF7CA5
Round 7 10AF9D37 308BEE97 02765708B5BF
Round 8 308BEE97 A9FC20A3 84BB4473DCCC
Round 9 A9FC20A3 2E8F9C65 34F822F0C66D
Round 10 2E8F9C65 A15A4B87 708AD2DDB3C0
Round 11 A15A4B87 236779C2 C1948E87475E
Round 12 236779C2 B8089591 69A629FEC913
Round 13 B8089591 4A1210F6 DA2D032B6EE3
Round 14 4A1210F6 5A78E394 06EDA4ACF5B5
Round 15 5A78E394 18CA18AD 4568581ABCCE
Round 16 18CA18AD 14A7D678 194CD072DE8C

Plain Text: 123456ABCD132536

Practical 7

Aim: Implement Diffi-Hellmen Key exchange Method.

Objective: The Diffie–Hellman key exchange method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure channel. The method was followed shortly afterwards by RSA, an implementation of public-key cryptography using asymmetric algorithms.

Theory:

Diffie Hellman (DH) key exchange algorithm is a method for securely exchanging cryptographic keys over a public communications channel. Keys are not actually exchanged – they are jointly derived. It is named after their inventors Whitfield Diffie and Martin Hellman.

If Alice and Bob wish to communicate with each other, they first agree between them a large prime number p , and a generator (or base) g (where $0 < g < p$).

Alice chooses a secret integer a (her private key) and then calculates $g^a \bmod p$ (which is her public key). Bob chooses his private key b , and calculates his public key in the same way.

Bob knows b and g^a , so he can calculate $(g^a)^b \bmod p = g^{ab} \bmod p$. Therefore both Alice and Bob know a shared secret $g^{ab} \bmod p$. An eavesdropper Eve who was listening in on the communication knows p , g , Alice's public key ($g^a \bmod p$) and Bob's public key ($g^b \bmod p$). She is unable to calculate the shared secret from these values.

In static-static mode, both Alice and Bob retain their private/public keys over multiple communications. Therefore the resulting shared secret will be the same every time. In ephemeral-static mode one party will generate a new private/public key every time, thus a new shared secret will be generated.

Program :

```
import java.io.*;
import java.math.BigInteger;

public class Main {
    public static void main(String[] args) throws IOException
    {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter prime number:");
        BigInteger p=new BigInteger(br.readLine());
        System.out.print("Enter primitive root of "+p+":");
        BigInteger g=new BigInteger(br.readLine());
        System.out.println("Enter value for x less than "+p+":");
        BigInteger x=new BigInteger(br.readLine());
        BigInteger R1=g.modPow(x,p);
        System.out.println("R1="+R1);
        System.out.print("Enter value for y less than "+p+":");
        BigInteger y=new BigInteger(br.readLine());
        BigInteger R2=g.modPow(y,p);
        System.out.println("R2="+R2);
        BigInteger k1=R2.modPow(x,p);
        System.out.println("Key calculated at Alice's side:"+k1);
        BigInteger k2=R1.modPow(y,p);
        System.out.println("Key calculated at Bob's side:"+k2);
        System.out.println("diffie hellman secret key Encryption has Taken");
    }
}
```

}

Output :

```
Enter prime number:
23
Enter primitive root of 23:5
Enter value for x less than 23:
7
R1=17
Enter value for y less than 23:8
R2=16
Key calculated at Alice's side:18
Key calculated at Bob's side:18
deffie hellman secret key Encryption has Taken

...Program finished with exit code 0
Press ENTER to exit console.
```

Practical 8

Aim: Implement RSA encryption-decryption algorithm.

Objective: RSA (Rivest–Shamir–Adleman) 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 the keys can be given to anyone.

Theory:

RSA algorithm is asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. Public Key and Private Key. As the name describes that the Public Key is given to everyone and Private key is kept private.

An example of asymmetric cryptography :

A client (for example browser) sends its public key to the server and requests for some data. The server encrypts the data using client's public key and sends the encrypted data. Client receives this data and decrypts it. Since this is asymmetric, nobody else except browser can decrypt the data even if a third party has public key of browser.

RSA:

The idea! The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key consists of two numbers where one number is multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So if somebody can factorize the large number, the private key is compromised. Therefore encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially. RSA keys can be typically 1024 or 2048 bits long, but experts believe that 1024 bit keys could be broken in the near future. But till now it seems to be an infeasible task.

Program :

```
import java.io.DataInputStream;
import java.io.IOException;
import java.math.BigInteger;
import java.util.Random;

public class Main
{
    private BigInteger p;
    private BigInteger q;
    private BigInteger N;
    private BigInteger phi;
    private BigInteger e;
    private BigInteger d;
    private int    bitlength = 1024;
    private Random  r;
    public Main()
    {
        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 Main(BigInteger e, BigInteger d, BigInteger N)
{
    this.e = e;
    this.d = d;
    this.N = N;
}
@SuppressWarnings("deprecation")
public static void main(String[] args) throws IOException
{
    Main rsa = new Main();
    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()));
    byte[] encrypted = rsa.encrypt(teststring.getBytes());
    byte[] decrypted = rsa.decrypt(encrypted);
    System.out.println("Decrypting Bytes: " + bytesToString(decrypted));
    System.out.println("Decrypted String: " + new String(decrypted));
}
private static String bytesToString(byte[] encrypted)
{

```

```
String test = "";
for (byte b : encrypted)
{
    test += Byte.toString(b);
}
return test;
}

public byte[] encrypt(byte[] message)
{
    return (new BigInteger(message)).modPow(e, N).toByteArray();
}

public byte[] decrypt(byte[] message)
{
    return (new BigInteger(message)).modPow(d, N).toByteArray();
}
}
```

Output :

```
Enter the plain text:
iampronooob
Encrypting String: iampronooob
String in Bytes: 1059710911211411111011111198
Decrypting Bytes: 1059710911211411111011111198
Decrypted String: iampronooob

...Program finished with exit code 0
Press ENTER to exit console.█
```

Practical – 9

Aim: Write a program to generate SHA-1 hash.

Theory:

SHA-1 or Secure Hash Algorithm 1 is a cryptographic hash function which takes an input and produces a 160-bit (20-byte) hash value. This hash value is known as a message digest. This message digest is usually then rendered as a hexadecimal number which is 40 digits long. It is a U.S. Federal Information Processing Standard and was designed by the United States National Security Agency. SHA-1 is now considered insecure since 2005. Major tech giants' browsers like Microsoft, Google, Apple and Mozilla have stopped accepting SHA-1 SSL certificates by 2017.

Objective:

Generating a hash value of the given string using SHA-1 algorithm

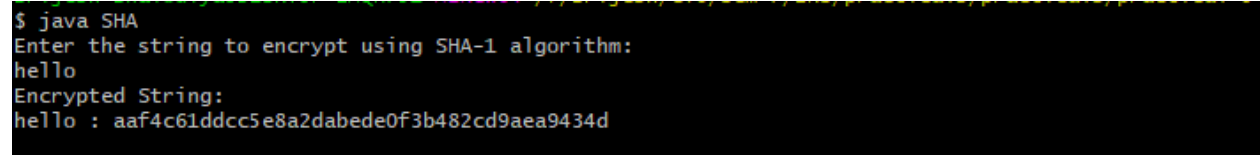
Program:

```
import java.util.Scanner;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class SHA {
    public static String encryptThisString(String input)
    {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-1");
            byte[] messageDigest = md.digest(input.getBytes());
            BigInteger no = new BigInteger(1, messageDigest);
            String hashtext = no.toString(16);
            while (hashtext.length() < 32) {
                hashtext = "0" + hashtext;
            }
        }
    }
}
```

```
        return hashtext;
    }
    catch (NoSuchAlgorithmException e) {
        throw new RuntimeException(e);
    }
}

public static void main(String args[]) throws NoSuchAlgorithmException
{
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the string to encrypt using SHA-1 algorithm: ");
    String str = sc.nextLine();
    System.out.println("Encrypted String:");
        System.out.println(str + " : " + encryptThisString(str));
    }
}
```

Output:

```
$ java SHA
Enter the string to encrypt using SHA-1 algorithm:
hello
Encrypted String:
hello : aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d
```

Practical – 10

Aim: Implement a digital signature algorithm.

Theory:

The Digital Signature Algorithm (DSA) is a Federal Information Processing Standard for digital signatures, based on the mathematical concept of modular exponentiation and the discrete logarithm problem.

The algorithm uses a key pair consisting of a public key and a private key. The private key is used to generate a digital signature for a message, and such a signature can be verified by using the signer's corresponding public key. The digital signature provides message authentication (the receiver can verify the origin of the message), integrity (the receiver can verify that the message has not been modified since it was signed) and non-repudiation (the sender cannot falsely claim that they have not signed the message).

Digital Signature Flow:

- Let “A” and “B” be the fictional actors in the cryptography system for better understanding.
- “A” is the sender and calculates the hash of the message and attaches signature which he wants to send using his private key.
- The other side “B” hashes the message and then decrypts the signature with A’s public key and compares the two hashes
- If “B” finds the hashes matching, then the message has not been altered or compromised.

Objective:

To generate a digital signature using algorithms SHA & RSA & also verify if the hash matches with a public key.

Program:

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.SecureRandom;
import java.security.Signature;
import java.util.Scanner;

public class DigitalSign {

    private final static char[] hexArray = "0123456789ABCDEF".toCharArray();
    private static final String SIGNING_ALGORITHM = "SHA256withRSA";
    private static final String RSA = "RSA";
    private static Scanner sc;

    public static byte[] Create_Digital_Signature(byte[] input, PrivateKey Key) throws
    Exception {
        Signature signature = Signature.getInstance(SIGNING_ALGORITHM);
        signature.initSign(Key);
        signature.update(input);
        return signature.sign();
    }

    public static KeyPair Generate_RSA_KeyPair() throws Exception {
        SecureRandom secureRandom = new SecureRandom();
        KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance(RSA);
        keyPairGenerator.initialize(2048, secureRandom);
        return keyPairGenerator.generateKeyPair();
    }
}
```

```
}
```

```
public static boolean Verify_Digital_Signature(byte[] input,byte[]
signatureToVerify,PublicKey key) throws Exception {
    Signature signature = Signature.getInstance(SIGNING_ALGORITHM);
    signature.initVerify(key);
    signature.update(input);
    return signature.verify(signatureToVerify);
}
```

```
public static String bytesToHex(byte[] bytes) {
    char[] hexChars = new char[bytes.length * 2];
    for ( int j = 0; j < bytes.length; j++ ) {
        int v = bytes[j] & 0xFF;
        hexChars[j * 2] = hexArray[v >>> 4];
        hexChars[j * 2 + 1] = hexArray[v & 0x0F];
    }
    return new String(hexChars);
}
```

```
public static void main(String args[]) throws Exception {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the String to encrypt");
    String input = sc.nextLine();
    KeyPair keyPair = Generate_RSA_KeyPair();

    byte[] signature = Create_Digital_Signature(input.getBytes(), keyPair.getPrivate());

    System.out.println("Signature Value:\n " + bytesToHex(signature));
}
```



```
        System.out.println("Verified: " + Verify_Digital_Signature(input.getBytes(), signature,
keyPair.getPublic()));
    }
}
```

Output:

```
$ java DigitalSign
Enter the String to encrypt
Hello World
Signature Value:
63C85794C74316926A6164EF0DAC618BD8293D0F38F9FC27DB946B681854CADA22E15E3624995229D39B1E4A0C2F33694168E52C
45FCA96E73DBD6BA0AC78283B3C35C4B3CAED4C7FB3B243BFEEA780FF997638B19179FE5FD073FDAEEC5159A631F1A6BACA214AC0
CCD5CB02F8DC5DEB91130DBD732B594E5EB1DE9DCA73AB7C1327CB41DF30699AC0991C6EC484B5122920AA6692AC0B81032F89559
71A14CE48E719CD386E29FB113027DA876256515049950ECFD81D51AAB43C6501058E2F58C26DD854D6E96B896E1DC125D59732AA
F5421792099619C46AA09829F19FA4E7025478335958F407DC3E6ED405DED3D11A0D856A81047A9655FFA288B768D
Verified: true
```

Practical – 11

Aim: Perform various encryption-decryption techniques with cryptool.

Theory:

Cryptool is an open-source and freeware program that can be used in various aspects of cryptographic and cryptanalytic concepts.

What is Cryptool?

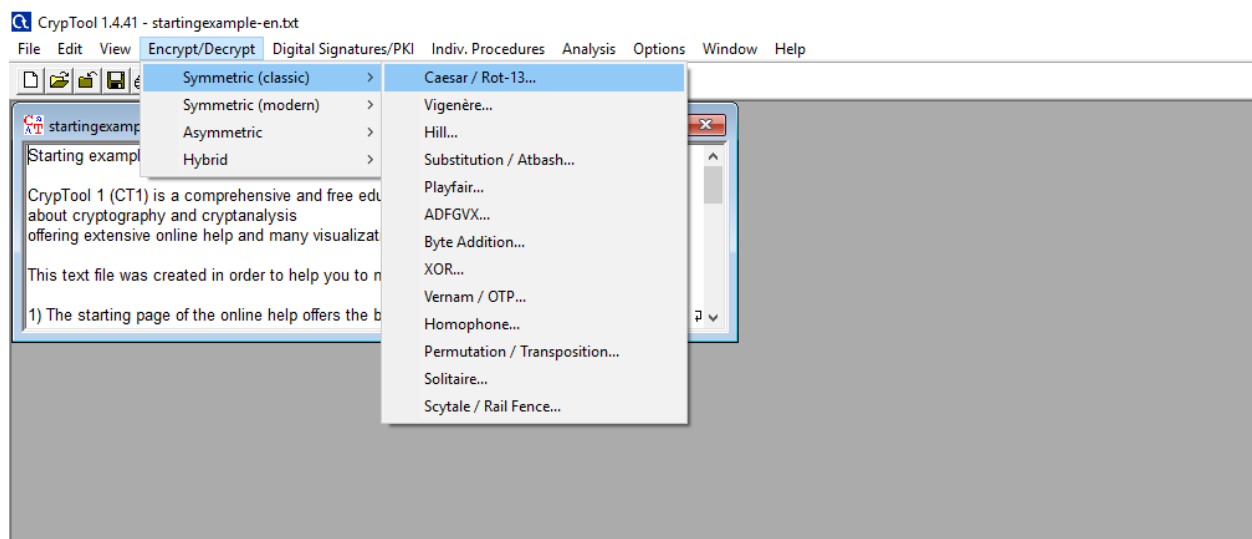
- A freeware program with graphical user interface (GUI).
- A tool for applying and analyzing cryptographic algorithms.
- With extensive online help, it's understandable without deep crypto knowledge.
- Contains nearly all state-of-the-art crypto algorithms.
- “Playful” introduction to modern and classical cryptography.
- Not a “hacker” tool.

Objective:

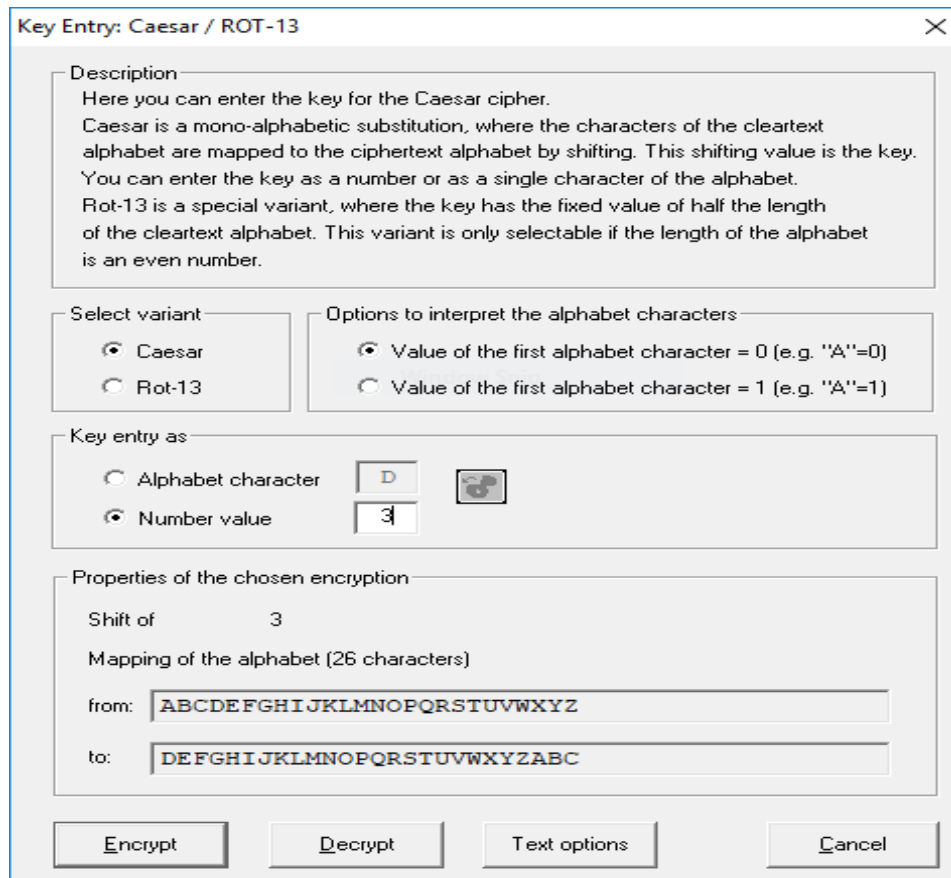
Perform Caesar cipher, Playfair cipher, Hill cipher, Vigenere Cipher in cryptool.

Encryption and Decryption of Caesar Cipher:

- We will implement an encryption and decryption of Caesar Cipher, which is actually a substitution method of cryptography. The Caesar Cipher involves replacing each letter of the alphabet with a letter – placed down or up according to the key given.
- To start with the process you have to move to the Encrypt/Decrypt tab of the program. There, you will find Symmetric (Classic) tab - Choose Caesar Cipher.



- In encryption, we are replacing the plaintext letter with the 3rd letter of the alphabet that is if "A" is our plaintext character, then the Ciphertext will be "D".




Key Entry: Caesar / ROT-13

Description
Here you can enter the key for the Caesar cipher.
Caesar is a mono-alphabetic substitution, where the characters of the cleartext alphabet are mapped to the ciphertext alphabet by shifting. This shifting value is the key. You can enter the key as a number or as a single character of the alphabet.
Rot-13 is a special variant, where the key has the fixed value of half the length of the cleartext alphabet. This variant is only selectable if the length of the alphabet is an even number.

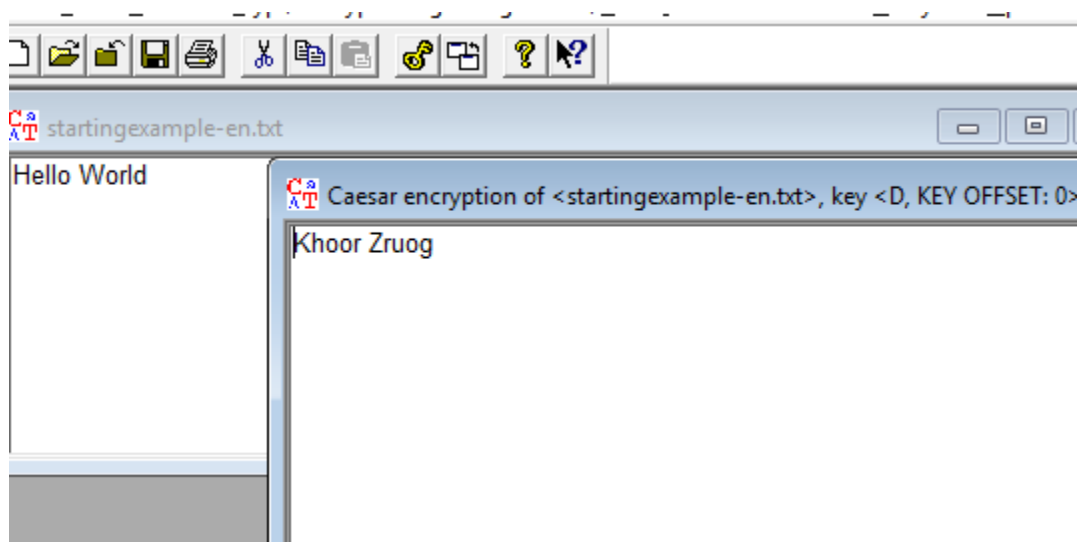
Select variant
☒ Caesar
☐ Rot-13

Options to interpret the alphabet characters
☒ Value of the first alphabet character = 0 (e.g. "A"=0)
☐ Value of the first alphabet character = 1 (e.g. "A"=1)

Key entry as
☐ Alphabet character 
☒ Number value

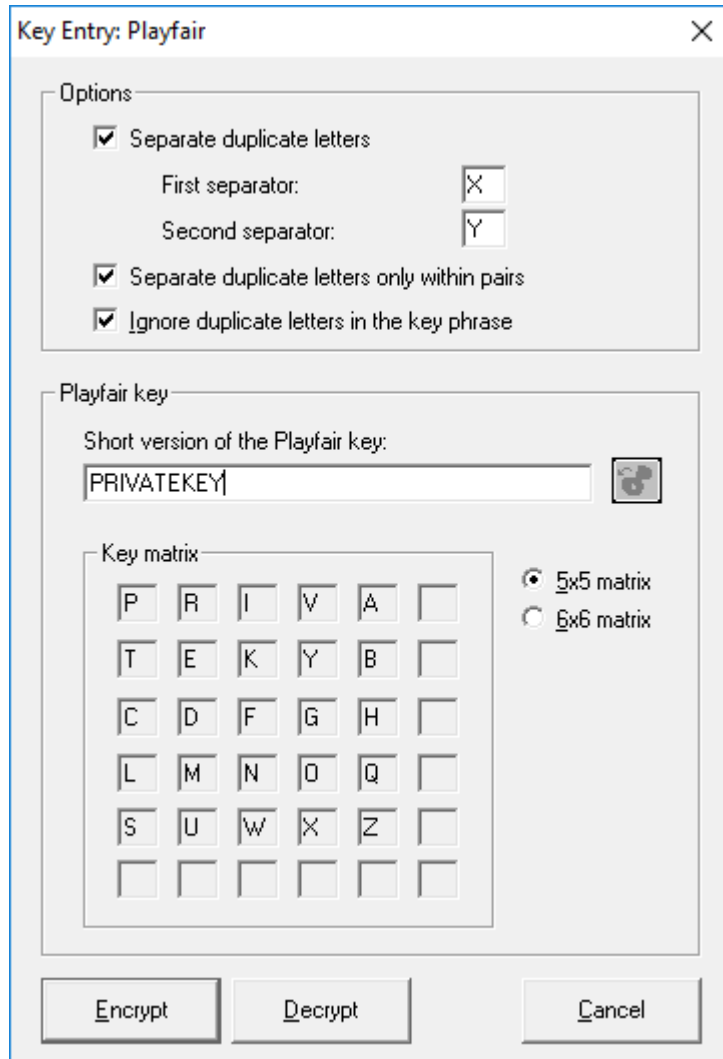
Properties of the chosen encryption
Shift of 3
Mapping of the alphabet (26 characters)
from: ABCDEFGHIJKLMNOPQRSTUVWXYZ
to: DEFPGHIJKLMNOPQRSTUVWXYZABC

- So, if we give "Hello World" as plaintext in Caesar Cipher, it will show me the encryption, as shown in the below image.




Encryption and Decryption of Playfair:

- Again, we have to move to Encrypt/Decrypt - Symmetric - Playfair Cipher and perform the encryption part. We are putting the same plaintext – “Hello World” & for Playfair key we are going to use “PRIVATEKEY” as a key.

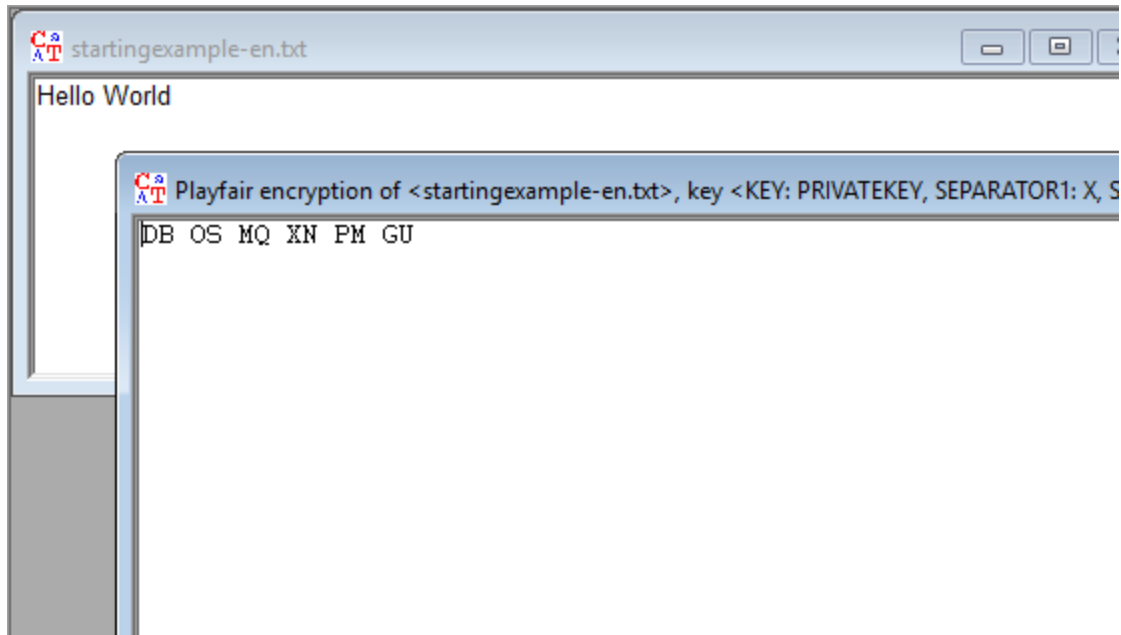


The image shows a 'Key Entry: Playfair' dialog box with the following components:

- Options:**
 - ☒ Separate duplicate letters
 - First separator:
 - Second separator:
 - ☒ Separate duplicate letters only within pairs
 - ☒ Ignore duplicate letters in the key phrase
- Playfair key:**
 - Short version of the Playfair key: 
 - Key matrix:**

P	R	I	V	A	
T	E	K	Y	B	
C	D	F	G	H	
L	M	N	O	Q	
S	U	W	X	Z	
 - ☒ 5x5 matrix
☐ 6x6 matrix
- Buttons:**

- So, when we press the encrypt button, we will get the Cipher text – “DBOSMQXNPMGU”.



Encryption and Decryption of Hill Cipher:

- Again, we have to move to Encrypt/Decrypt - Symmetric - Hill Cipher and perform the encryption part. We are putting the plaintext as – Hello and assuming that the program gives us the Cipher text as – WuvjoD.

Key Entry: Hill

Description
The Hill cipher is a polygraphic substitution cipher based on linear algebra. This was the first polygraphic cipher in which it was practical to operate on groups of more than three letters (blocks) at once. The key is a quadratic matrix. Its dimension is the length of the group of letters.

Selected alphabet (26 characters)
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Value of the first alphabet character: 1

Hill key matrix
☐ Alphabet characters
☒ Number values
 Alphabet characters: A C
 Number values: 01 03
 02 01
 Generate random key
 Reset key

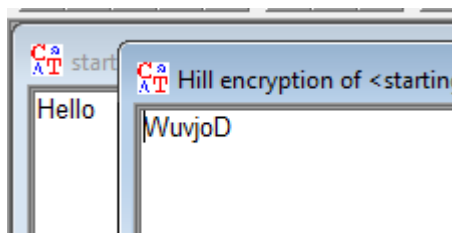
Multiplication variant
☐ (row vector) * (matrix)
☒ (matrix) * (column vector)

Size of matrix
☐ 1 x 1
☒ 2 x 2
☐ 3 x 3
☐ 4 x 4
☐ 5 x 5
 Larger matrix

☐ Show details and single steps of the Hill cipher

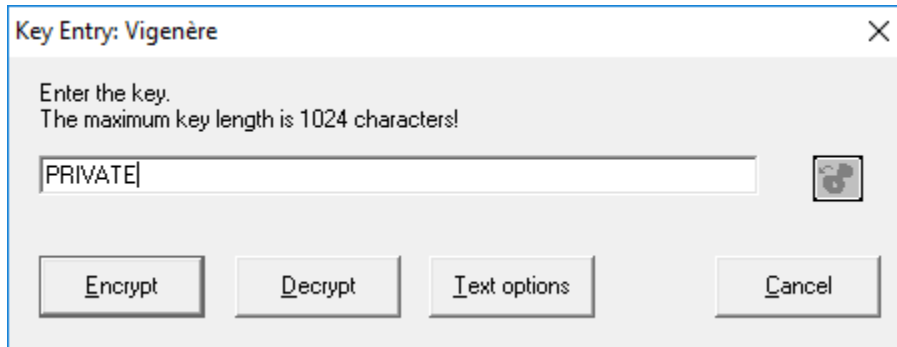
Encrypt Decrypt Further Hill options Text options Cancel

- So, when we press the encrypt button, we will get the Cipher text – “WuvjoD”.

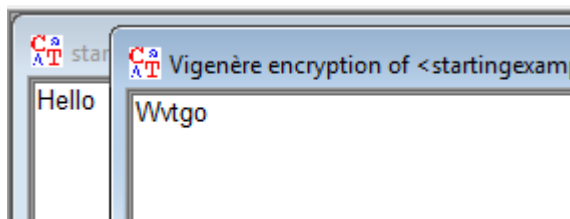


Encryption and Decryption of Vigenere Cipher:

- Again, we have to move to Encrypt/Decrypt - Symmetric - Vigenere Cipher and perform the encryption part. We are putting the plaintext as – HELLOWORLD and assuming that the program gives us the Cipher text as – Wvtgo, with the help of key as – PRIVATE.



- So, when we press the encrypt button, we will get the Cipher text somewhat like – “Wvtgo”.



Practical – 12

Aim: Study and use the Wireshark for the various network Protocols.

Theory:

- Wireshark is a network protocol analysis tool. At its core, Wireshark was designed to break down packets of data being transferred across different networks. The user can search and filter for specific packets of data and analyze how they are transferred across their network. These packets can be used for analysis on a real-time or offline basis.
- The user can use this information to generate statistics and graphs. Wireshark was originally known as Ethereal but has since established itself as one of the key network analysis tools on the market. This is the go-to tool for users who want to view data generated by different networks and protocols.
- Wireshark is suitable for novice and expert users alike. The user interface is incredibly simple to use once you learn the initial steps to capture packets. More advanced users can use the platform's decryption tools to break down encrypted packets as well.

Wireshark core features:

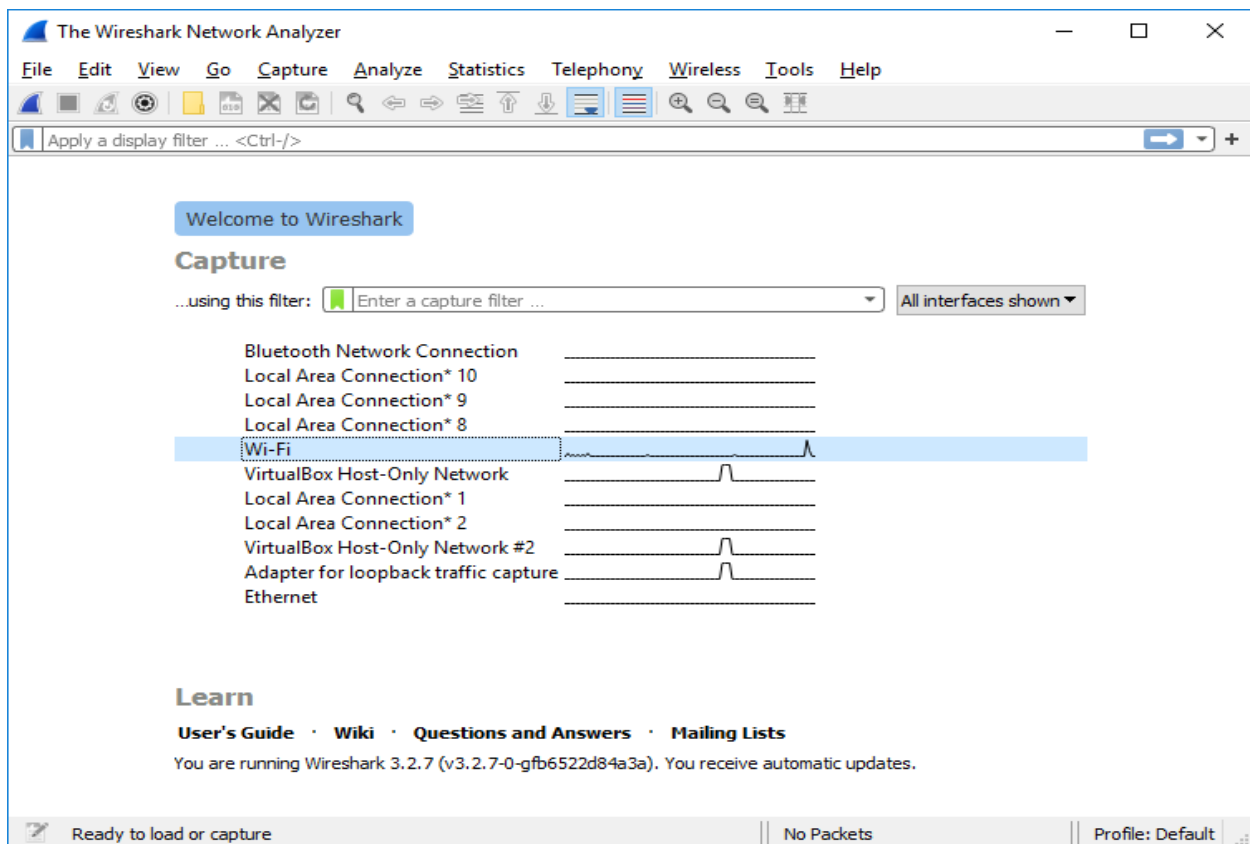
- Capture live packet data
- Import packets from text files
- View packet data and protocol information
- Save captured packet data
- Display packets
- Filter packets
- Search packets
- Colorize packets
- Generate Statistics

Objective:

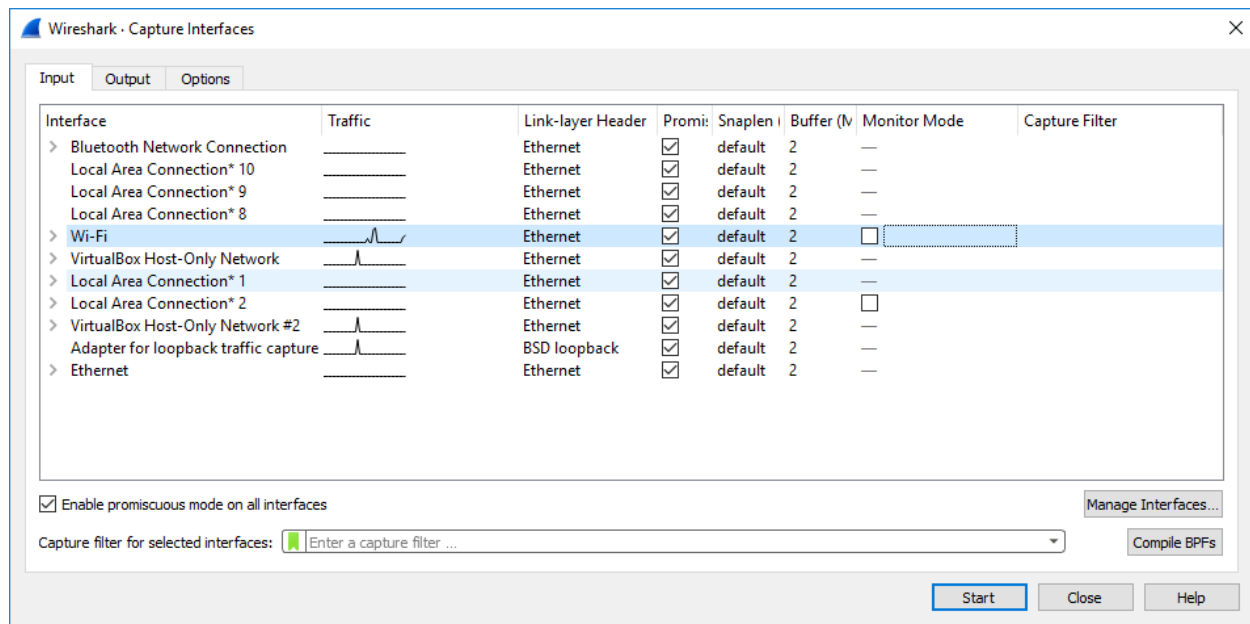
Analyze the data packets using the Wireshark.

Capture Data Packets:

- When you first open up Wireshark, you'll be met by the following launch screen:



- The first thing you need to do is look at the available interfaces to capture. To do this, select **Capture > Options**. The “Capture Interfaces” dialog box will then open as shown below:



- Click **start**, Wireshark start analyzing the packets.
- In the screenshot below there are three panes, the **packet list** pane, the **packet bytes** pane, and the **packet details** pane.

The screenshot shows the Wireshark interface with the title 'Capturing from Wi-Fi'. The menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. The toolbar contains various icons for file operations, capture control, and analysis. A filter bar at the top shows 'Apply a display filter ... <Ctrl-/>'. The packet list pane displays a table of captured packets:

No.	Time	Source	Destination	Protocol	Length	Info
771	73.773711	192.168.43.65	204.79.197.200	TCP	1214	50501 → 443 [ACK] Seq=276
772	73.778903	192.168.43.65	172.217.166.206	TCP	55	[TCP Keep-Alive] 50492 →
773	73.781919	204.79.197.200	192.168.43.65	TCP	54	443 → 50501 [ACK] Seq=606
774	73.781966	192.168.43.65	204.79.197.200	TLSv1.2	1026	Application Data, Applica
775	73.784899	204.79.197.200	192.168.43.65	TCP	54	443 → 50501 [ACK] Seq=606
776	73.784899	204.79.197.200	192.168.43.65	TCP	54	443 → 50501 [ACK] Seq=606
777	73.784899	204.79.197.200	192.168.43.65	TCP	54	443 → 50501 [ACK] Seq=606
778	73.834990	204.79.197.200	192.168.43.65	TCP	54	443 → 50501 [ACK] Seq=606
779	73.844737	204.79.197.200	192.168.43.65	TCP	54	443 → 50501 [ACK] Seq=606
780	73.847212	204.79.197.200	192.168.43.65	TCP	54	443 → 50501 [ACK] Seq=606
781	73.849814	204.79.197.200	192.168.43.65	TCP	54	443 → 50501 [ACK] Seq=606
782	73.855533	204.79.197.200	192.168.43.65	TCP	54	443 → 50501 [ACK] Seq=606
783	73.855533	204.79.197.200	192.168.43.65	TCP	54	443 → 50501 [ACK] Seq=606
784	73.882036	172.217.166.206	192.168.43.65	TCP	66	[TCP Keep-Alive ACK] 443
785	73.942221	204.79.197.200	192.168.43.65	TLSv1.2	198	Application Data
786	73.942362	192.168.43.65	204.79.197.200	TCP	54	50501 → 443 [ACK] Seq=297

Below the packet list, the details pane shows the selected packet (Frame 1) with the following information:

- Frame 1: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface \Device\NPF_{A6395515}
- Ethernet II, Src: IntelCor_26:6e:c0 (a0:d3:7a:26:6e:c0), Dst: 5a:76:ef:fe:68:2f (5a:76:ef:fe:68:2f)
- Internet Protocol Version 4, Src: 192.168.43.65, Dst: 192.168.43.23

The packet bytes pane at the bottom shows the raw data of the selected packet in hexadecimal and ASCII:

```

0000  5a 76 ef fe 68 2f a0 d3 7a 26 6e c0 08 00 45 00  5a 76 ef fe 68 2f a0 d3 7a 26 6e c0 08 00 45 00
0010  00 48 29 37 00 00 80 11 39 c5 c0 a8 2b 41 c0 a8  00 48 29 37 00 00 80 11 39 c5 c0 a8 2b 41 c0 a8
0020  2b 17 df b8 00 35 00 34 09 44 bd 62 01 00 00 01  2b 17 df b8 00 35 00 34 09 44 bd 62 01 00 00 01
  
```

The status bar at the bottom indicates 'Wi-Fi: <live capture in progress>' and 'Packets: 786 - Displayed: 786 (100.0%)'.

- For more packet information, click on any of the fields in each packet to see more. When you click on a packet, we can see a breakdown of its internal bytes in the byte view section.

Packet List:

The packet list pane is shown at the top of the screenshot. Each piece is broken down to a number with time, source, destination, protocol and support information.

Packet Details:

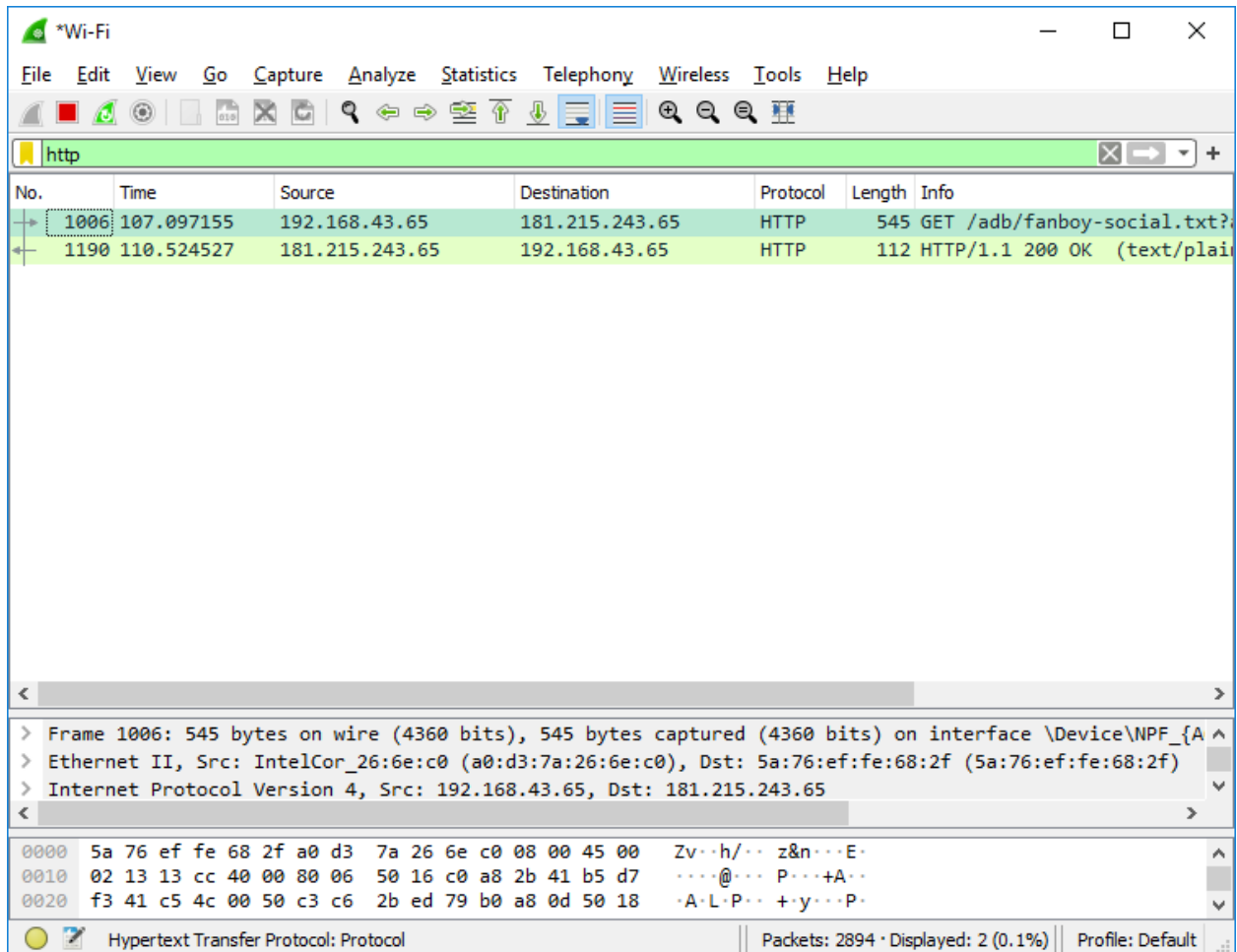
Packet details can be found in the middle, showing the protocols of the chosen packet. You can expand each section by clicking on the arrow next to your row of choice. You can also apply additional filters by right-clicking on the chosen item.

Packet Bytes:

The packet bytes pane is shown at the bottom of the page. This pane shows the internal data of your selected packet. If you highlight part of the data in this section,

its corresponding information is also highlighted in the packet details pane. By default, all data is shown in hexadecimal format. If you want to change it to bit format, right-click the pane and select this option from the context menu.

- To filter the packets according to protocols, add filter expression in filter input field as shown in the below figure.



- To create a visual representation of data packets, simply click on the statistics menu and select IO graphs which display graph as shown in the below image.

