DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Bawana Road, Delhi 110042

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CO-405 : Information and Network Security Lab File

Submitted To:
Dr. Shailender Kumar
Associate Professor
Department of Computer Science and Engineering

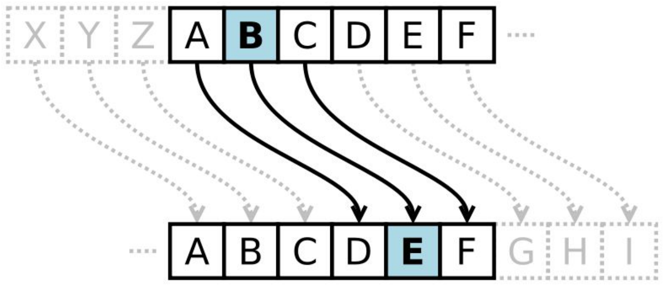
Submitted By:
Kunal Sinha
B.Tech Computer Science
7th Semester
2K17/CO/164

INDEX

Experiment 1

Aim: Write a program to implement Caesar Cipher encryption and decryption.

Theory: The Caesar Cipher technique is one of the earliest and simplest methods of encryption technique. It's simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter with 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. So to cipher a given text we need an integer value, known as shift which indicates the number of positions each letter of the text has been moved down.



Mathematically,

$$\text{Encryption} = En = (x+k) \; mod \; 26$$
$$\text{Decryption} = Dn = (x+ k-1) \; mod \; 26$$

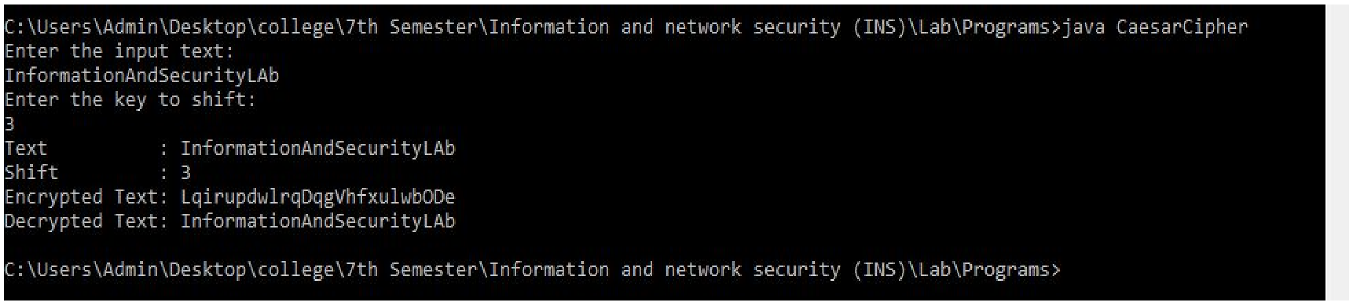
Where x is the input alphabet ASCII representation and k the key, k-1 is the additive inverse of k.

Source Code:

```
import java.util.Scanner;
public class CaesarCipher
{    public static String encryption(String text, int k)
   {     String result="";
       for (int i=0; i<text.length(); i++)
       {         if (Character.isUpperCase(text.charAt(i)))
          {    char ch = (char)(((int)text.charAt(i)+k-65)%26+65);
             result+=ch;
          }
else
          {
             char ch = (char)(((int)text.charAt(i)+k-97)%26+97);
             result+=ch;
          }
       }
```

```
        return result;
    }
    public static String decryption(String text, int k)
    {
        String result=encryption(text, 26-k);
        return result;
    }
    public static void main(String[] args)
    { Scanner myObj = new Scanner(System.in);
        System.out.println("Enter the input text:");
        String text = myObj.nextLine();
        System.out.println("Enter the key to shift:");
        int k= myObj.nextInt();
        String encrypt=encryption(text, k).toString();
        System.out.println("Text          : " + text);
        System.out.println("Shift         : " + k);
        System.out.println("Encrypted Text: " + encrypt);
        System.out.println("Decrypted Text: " + decryption(encrypt,k));
    }
}
```

Output:



```
C:\Users\Admin\Desktop\college\7th Semester\Information and network security (INS)\Lab\Programs>java CaesarCipher
Enter the input text:
InformationAndSecurityLAb
Enter the key to shift:
3
Text          : InformationAndSecurityLAb
Shift         : 3
Encrypted Text: LqirupdwlrqDqgVhfxulwbODe
Decrypted Text: InformationAndSecurityLAb

C:\Users\Admin\Desktop\college\7th Semester\Information and network security (INS)\Lab\Programs>
```

Learning Outcomes:

The Caesar cipher offers essentially no communication security and it can be easily broken.

Experiment 2

Aim: Write a program to implement Monoalphabetic encryption and decryption.

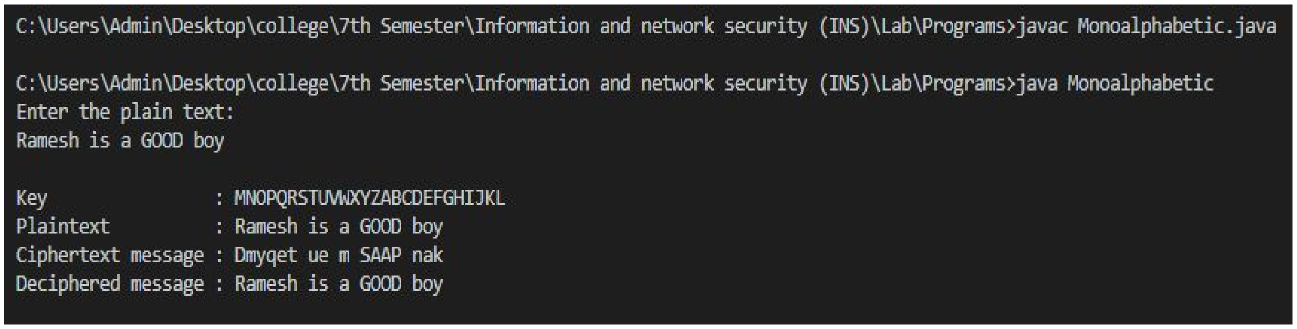
Theory: Monoalphabetic cipher is a substitution cipher in which for a given key, the cipher alphabet for each plain alphabet is fixed throughout the encryption process. For example, if 'A' is encrypted as 'D', for any number of occurrences in that plaintext, 'A' will always get encrypted to 'D'. The possible number of keys is large (26!) and so Monoalphabetic ciphers are highly susceptible to cryptanalysis.

Source Code:

```
import java.util.Scanner;
public class Monoalphabetic {

  private static final String alphabet = "abcdefghijklmnopqrstuvwxyz";
  private static final String Al = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

  public static String encrypt(String plaintext, String key) {
    String ciphertext = "";

    for (char chr : plaintext.toCharArray()) {
      if(Character.isUpperCase(chr))
      {
      byte position = (byte) Al.indexOf(chr);
      if (chr == ' ') {
          ciphertext+=" ";
        }
      else {
          ciphertext+=Character.toUpperCase(key.charAt(position));
          }
      }
      else
      {
        byte position = (byte) alphabet.indexOf(chr);
        if (chr == ' ') {
            ciphertext+=" ";
          }
        else {
                ciphertext+=key.charAt(position);
              }
        }
    }
    return ciphertext;
}

public static String decrypt(String ciphertext, String key) {
    String plaintext ="";
    for (char chr : ciphertext.toCharArray()) {
        if(Character.isUpperCase(chr))
        {
        byte position = (byte) key.indexOf(Character.toLowerCase(chr));
        if (chr == ' ') {
            plaintext+=" ";
        }
        else {
            plaintext+=Al.charAt(position);
            }
        }
        else
        {
            byte position = (byte) key.indexOf(chr);
            if (chr == ' ') {
                plaintext+=" ";
            }
            else {
                plaintext+=alphabet.charAt(position);
                }
        }
    }
    return plaintext;
}

public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);
    System.out.println("Enter the plain text:");
    String text = myObj.nextLine();
    String key="mnopqrstuvwxyzabcdefghijkl";
```

```
        System.out.println("\nKey              : "+key.toUpperCase()+"\nPlaintext        : "+text );
        String ciphertext=encrypt(text, key);
        System.out.println("Ciphertext message : "+ciphertext);
        System.out.println("Deciphered message : " +decrypt(ciphertext,key ));
    }
}
```

Output:



```
C:\Users\Admin\Desktop\college\7th Semester\Information and network security (INS)\Lab\Programs>javac Monoalphabetic.java

C:\Users\Admin\Desktop\college\7th Semester\Information and network security (INS)\Lab\Programs>java Monoalphabetic
Enter the plain text:
Ramesh is a GOOD boy

Key                : MNOPQRSTUVWXYZABCDEFGHIJKL
Plaintext          : Ramesh is a GOOD boy
Ciphertext message : Dmyqet ue m SAAP nak
Deciphered message : Ramesh is a GOOD boy
```

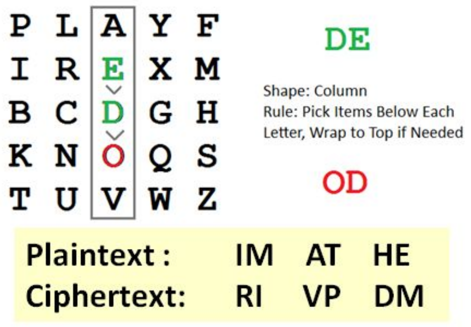
Learning Outcomes:

The modern computing systems are not yet powerful enough to comfortably launch a brute force attack to break the system having these ciphers. However, the Monoalphabetic Cipher has a simple design and it is prone to design flaws, say choosing obvious permutation, this cryptosystem can be easily broken.

Experiment 3

Aim: Write a program to implement Play-Fair Cipher encryption-decryption.

Theory: Playfair cipher, also known as Playfair square, Wheatstone-Playfair cipher or Wheatstone cipher is a manual symmetric encryption technique and was the first literal diagram substitution cipher. The scheme was invented in 1854 by Charles Wheatstone but was named after Lord Playfair who promoted the use of the cipher. In this scheme, pairs of letters are encrypted, instead of single letters as in the case of simple substitution ciphers.
In the Playfair cipher, initially a key table is created. The key table 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 we need only 25 alphabets instead of 26. If the plaintext contains J, then it is replaced by I. The sender and the receiver decide on a particular key, say 'Algorithm'. In a key table, the first characters (going left to right) in the table is the phrase, excluding the duplicate letters. The rest of the table will be filled with the remaining letters of the alphabet, in natural order.



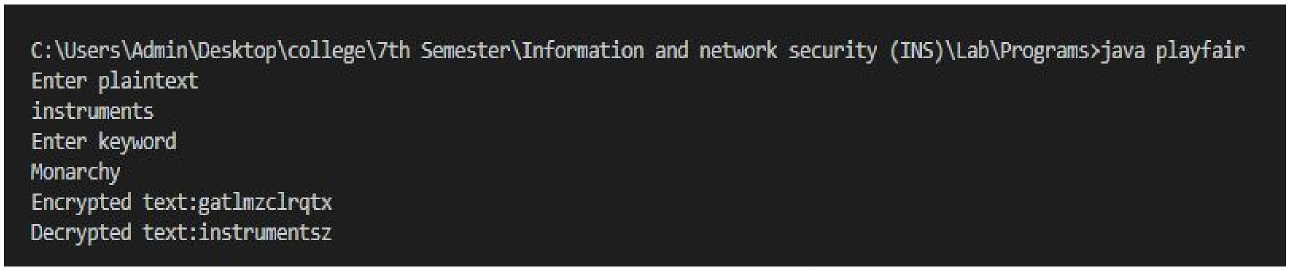
Source Code:

```
import java.util.*;
import java.util.Scanner;
public class playfair {
   public static void main(String[] args) {
      final Scanner sc = new Scanner(System.in);
      System.out.println("Enter plaintext");
         final String plaintext = sc.nextLine();
         System.out.println("Enter keyword");
         final String keyword = sc.nextLine();
         final String encryptedText = encrypt(plaintext, keyword);
         final String ptext = decrypt(encryptedText, keyword);
         System.out.println("Plain text    :"+plaintext);
```

```
          System.out.println("Keyword       :"+keyword);
          System.out.println("Encrypted text:"+encryptedText);
          System.out.println("Decrypted text:"+ptext);
        sc.close();
    }

    public static String decrypt(String encryptedText, String keyword) {
        if (encryptedText.length() % 2 != 0)
            {   throw new IllegalArgumentException("Encrypted text length must be even");   }
        encryptedText = encryptedText.toLowerCase();
        keyword = keyword.toLowerCase();
        final char[][] grid = generateGrid(keyword);
        final StringBuilder plaintext = new StringBuilder();
        for (int i = 0; i < encryptedText.length(); i += 2) {
            final char ch1 = encryptedText.charAt(i);
            final char ch2 = encryptedText.charAt(i + 1);
            final Position ch1Position = findPosition(grid, ch1);
            final Position ch2Position = findPosition(grid, ch2);
            final char plaintextCh1;
            final char plaintextCh2;
            if (ch1Position.column == ch2Position.column) {
            plaintextCh1 = grid[Math.floorMod(ch1Position.row -1,
grid.length)][ch1Position.column];
                plaintextCh2 = grid[Math.floorMod(ch2Position.row - 1,
grid.length)][ch2Position.column];
            } else if (ch1Position.row == ch2Position.row) {
                plaintextCh1 = grid[ch1Position.row][Math.floorMod(ch1Position.column - 1,
grid.length)];
                plaintextCh2 = grid[ch2Position.row][Math.floorMod(ch2Position.column - 1,
grid.length)];
            } else {
                final Position topRight;
                final Position bottomLeft;
                if (ch1Position.row < ch2Position.row) {
                    topRight = ch1Position;
                    bottomLeft = ch2Position;
                } else {
                    topRight = ch2Position;
                    bottomLeft = ch1Position;}
                final int rectWidth = topRight.column - bottomLeft.column + 1;
                final Position topLeft = new Position(topRight.row, topRight.column - rectWidth + 1);
                final Position bottomRight = new Position(bottomLeft.row, bottomLeft.column +
rectWidth - 1);
                if (ch1Position == topRight) {
                    plaintextCh1 = grid[topLeft.row][topLeft.column];
                    plaintextCh2 = grid[bottomRight.row][bottomRight.column];
                } else {
                    plaintextCh1 = grid[bottomRight.row][bottomRight.column];
                    plaintextCh2 = grid[topLeft.row][topLeft.column];
                }
            }
            plaintext.append(plaintextCh1).append(plaintextCh2);
        }
        return plaintext.toString();
    }

    public static String encrypt(String plaintext, String keyword) {
        plaintext = plaintext.toLowerCase();
        keyword = keyword.toLowerCase();
        final char[][] grid = generateGrid(keyword);
        final String diagraphText = plaintext.length() % 2 == 0 ? plaintext : plaintext + "z";
        final StringBuilder encryptedText = new StringBuilder();
        for (int i = 0; i < diagraphText.length(); i += 2) {
            final char ch1 = diagraphText.charAt(i);
            final char ch2 = diagraphText.charAt(i + 1);
            final Position ch1Pos = findPosition(grid, ch1);
            final Position ch2Pos = findPosition(grid, ch2);
            final char encryptedCh1;
            final char encryptedCh2;
            if (ch1Pos.column == ch2Pos.column) {
                encryptedCh1 = grid[Math.floorMod(ch1Pos.row + 1, grid.length)][ch1Pos.column];
                encryptedCh2 = grid[Math.floorMod(ch2Pos.row + 1, grid.length)][ch2Pos.column];
            } else if (ch1Pos.row == ch2Pos.row) {
                encryptedCh1 = grid[ch1Pos.row][Math.floorMod(ch1Pos.column + 1, grid.length)];
                encryptedCh2 = grid[ch2Pos.row][Math.floorMod(ch2Pos.column + 1, grid.length)];
            } else {
                final Position topLeft;
                final Position bottomRight;
                if (ch1Pos.row < ch2Pos.row) {
                    topLeft = ch1Pos;
                    bottomRight = ch2Pos;
                } else {
                    topLeft = ch2Pos;
                    bottomRight = ch1Pos;
                }
                final int rectWidth = bottomRight.column - topLeft.column + 1;
                final Position topRight = new Position(topLeft.row, topLeft.column + rectWidth - 1);
                final Position bottomLeft = new Position(bottomRight.row, bottomRight.column -
rectWidth + 1);
                if (ch1Pos == topLeft) {
                    encryptedCh1 = grid[topRight.row][topRight.column];
                    encryptedCh2 = grid[bottomLeft.row][bottomLeft.column];
                } else {
                    encryptedCh1 = grid[bottomLeft.row][bottomLeft.column];
                    encryptedCh2 = grid[topRight.row][topRight.column];   }
            }
            encryptedText.append(encryptedCh1).append(encryptedCh2);
        }
        return encryptedText.toString();
    }

    private static Position findPosition(char[][] grid, char ch) {
        for (int i = 0; i < grid.length; i++) {
            for (int j = 0; j < grid.length; j++) {
                if (grid[i][j] == ch)
                    {       return new Position(i, j);  }
            }}
        throw new IllegalArgumentException("Character " + ch + " not found in the grid");
    }

    private static char[][] generateGrid(String keyword) {
        final char[][] grid = new char[5][5];
        final Set<Character> charset = buildCharset(keyword);
        final Iterator<Character> charIterator = charset.iterator();
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                final char ch = charIterator.next();
                grid[i][j] = ch;  }
        }
        return grid;
    }

    private static Set<Character> buildCharset(String keyword) {
        final Set<Character> charset = new LinkedHashSet<>(25);
        for (final char ch : keyword.toCharArray()) {
            charset.add(ch);
        }
        final Set<Character> skipChars = new HashSet<>();
        if (charset.contains('j')) {
            if (!charset.contains('i')) {
                skipChars.add('i');
            }
        } else {
            skipChars.add('j');
        }
        for (char ch = 'a'; ch <= 'z' && charset.size() < 25; ch++) {
            if (!charset.contains(ch) && !skipChars.contains(ch)) {
                charset.add(ch);
            }
        }
        return charset;
    }

    private static class Position {
        final int row;
        final int column;
        Position(int row, int column) {
            this.row = row;
            this.column = column;
        }
        @Override
        public String toString() {
            return "(" + this.row + ", " + this.column + ")";
        }
```

```
        @Override
        public boolean equals(Object obj) {
            if (obj == this)
                        {  return true;}
            if (!(obj instanceof Position))
                        {  return false; }
            final Position other = (Position) obj;
            return this.row == other.row && this.column == other.column;
        }
        @Override
        public int hashCode() {
            return Objects.hash(this.row, this.column);
        }}
}
```

Output:



```
C:\Users\Admin\Desktop\college\7th Semester\Information and network security (INS)\Lab\Programs>java playfair
Enter plaintext
instruments
Enter keyword
Monarchy
Encrypted text:gatlmzclrqtx
Decrypted text:instrumentsz
```

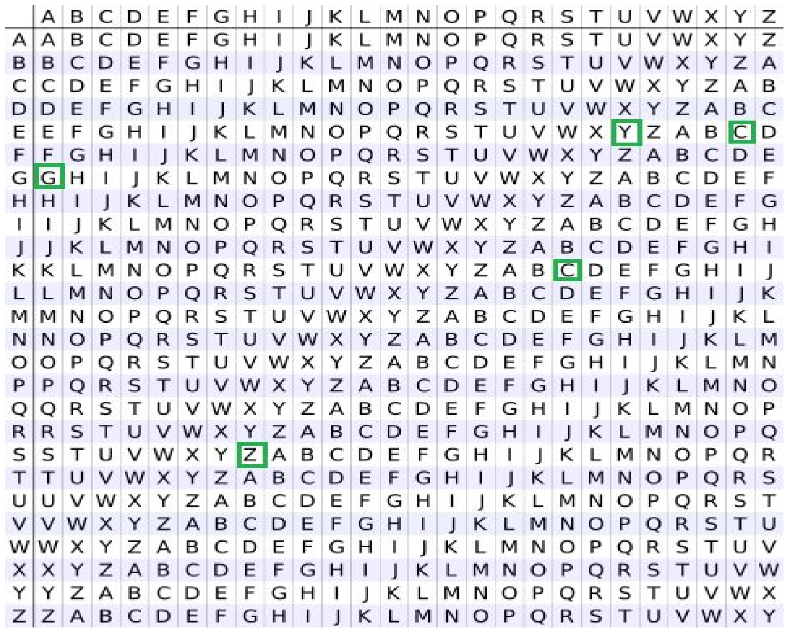
Learning Outcomes:

It is significantly harder to break since the frequency analysis technique used to break simple substitution ciphers is difficult but still can be used on (25*25) = 625 digraphs rather than 25 monographs which is difficult. Frequency analysis thus requires more cipher text to crack the encryption.

Experiment 4

Aim: Write a program to implement Polyalphabetic Cipher encryption-decryption.

Theory: In a polyalphabetic cipher, multiple cipher alphabets are used. To facilitate encryption, all the alphabets are usually written out in a large table, traditionally called a tableau. Usually the tableau is 26 × 26, so that 26 full ciphertext alphabets are available. The method of filling the tableau, and of choosing which alphabet to use next, defines the particular polyalphabetic cipher. All such ciphers are easier to break than were believed since the substitution alphabets are repeated for sufficiently large plaintexts. One of the most popular was that of Vigenere cipher.



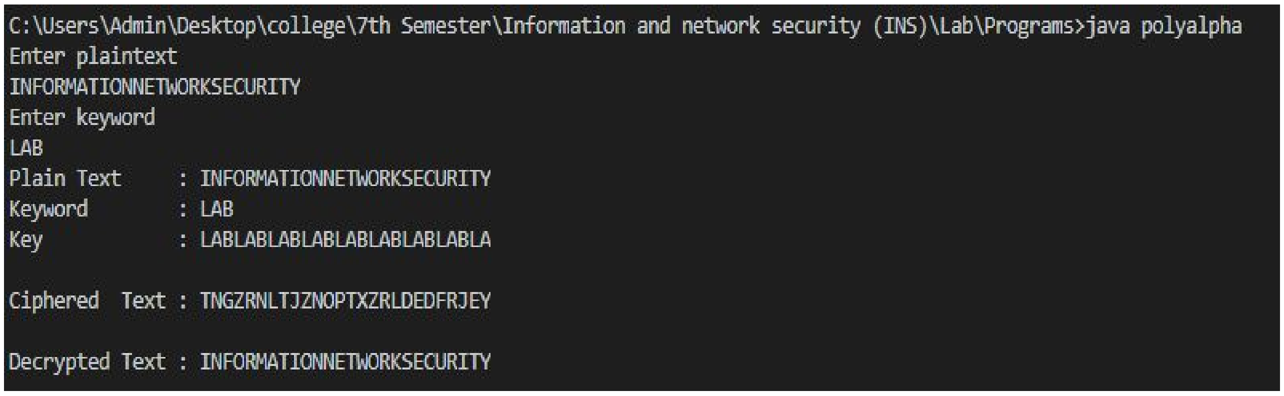
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	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

Source Code:

```
import java.util.*;
import java.util.Scanner;
public class polyalpha
{
static String generateKey(String str, String key)
{
   int x = str.length();
   for (int i = 0; ; i++)
   {
      if (x == i)
```

```
            i = 0;
        if (key.length() == str.length())
            break;
        key+=(key.charAt(i));
    }
    return key;
}

static String cipherText(String str, String key)
{
    String cipher_text="";
    for (int i = 0; i < str.length(); i++)
    {
        int x = (str.charAt(i) + key.charAt(i)) %26;
        x += 'A';
        cipher_text+=(char)(x);
    }
    return cipher_text;
}

static String originalText(String cipher_text, String key)
{
    String orig_text="";
    for (int i = 0 ; i < cipher_text.length() && i < key.length(); i++)
    {
        int x = (cipher_text.charAt(i) - key.charAt(i) + 26) %26;
        x += 'A';
        orig_text+=(char)(x);
    }
    return orig_text;
}

public static void main(String[] args)
    {
    final Scanner sc = new Scanner(System.in);
    System.out.println("Enter plaintext");
    final String str = sc.nextLine();
    System.out.println("Enter keyword");
    final String keyword = sc.nextLine();
    String key = generateKey(str, keyword);
    String cipher_text = cipherText(str, key);
    System.out.println("Ciphertext : " + cipher_text + "\n");
    System.out.println("Decrypted Text : " + originalText(cipher_text, key));
    sc.close();
    }
}
```

Output:



```
C:\Users\Admin\Desktop\college\7th Semester\Information and network security (INS)\Lab\Programs>java polyalpha
Enter plaintext
INFORMATIONNETWORKSECURITY
Enter keyword
LAB
Plain Text     : INFORMATIONNETWORKSECURITY
Keyword        : LAB
Key            : LABLABLABLABLABLABLABLABLA

Ciphered  Text : TNGZRNLTJZNOPTXZRLDEDFRJEY

Decrypted Text : INFORMATIONNETWORKSECURITY
```

Learning Outcomes:

The advantage of Polyalphabetic ciphers is that they make frequency analysis more difficult. For instance if P occurred most in a ciphertext whose plaintext is in English, one could suspect that P corresponded to E, because E is the most frequently used letter in English. Using the Polyalphabetic cipher, E can be enciphered as any of several letters in the alphabet in the cipher, thus defeating simple frequency analysis.

Experiment 5

Aim: Write a program to implement Hill Cipher encryption-decryption.

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 × 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 × n matrices (modulo 26)

Encryption

In order to encrypt a message using the Hill cipher, the sender and receiver must first agree upon a key matrix A of size n x n. A must be invertible mod 26. The plaintext will then be enciphered in blocks of size n. In the following example A is a 2 x 2 matrix and the message will be enciphered in blocks of 2 characters.

$$Encrypt(\text{ACT}) = POH$$

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix} = \begin{bmatrix} 67 \\ 222 \\ 319 \end{bmatrix} \equiv \begin{bmatrix} 15 \\ 14 \\ 7 \end{bmatrix} \%26$$

Decryption

Decryption follows Encryption but uses the inverse of the encryption matrix instead.

$$Decrypt(\text{"POH"}) = ACT$$

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix}^{-1} \begin{bmatrix} 15 \\ 14 \\ 7 \end{bmatrix} = \begin{bmatrix} 260 \\ 574 \\ 539 \end{bmatrix} \equiv \begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix} \%26$$

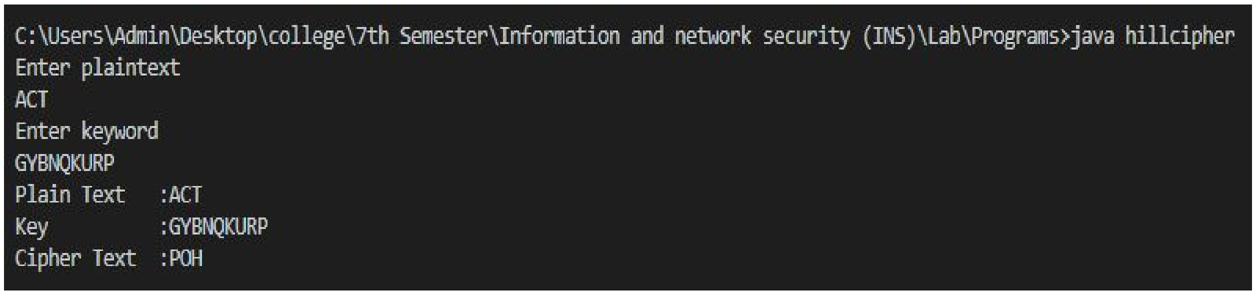
Source Code:

```
import java.util.*;
import java.util.Scanner;
public class hillcipher {
public static void getKeyMatrix(String key, int keyMatrix[][])
{
   int k = 0;
```

```
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            keyMatrix[i][j] = (key.charAt(k)) % 65;
            k++;
        }
    }
}
public static void encrypt(int cipherMatrix[][],int keyMatrix[][], int messageVector[][])
{
    int x, i, j;
    for (i = 0; i < 3; i++)
    { for (j = 0; j < 1; j++)
        {
            cipherMatrix[i][j] = 0;
            for (x = 0; x < 3; x++)
            {
                cipherMatrix[i][j] += keyMatrix[i][x] * messageVector[x][j];
            }
            cipherMatrix[i][j] = cipherMatrix[i][j] % 26;
        }
    }
}

public static void Decrypt(int decrypt[][],int keyMatrix[][], int messageVector[][])
{
    int i, j, k;
    int [][]b = new int[3][3];
    inverse(b);
    for(i = 0; i < 3; i++)
        {
            for(j = 0; j < 1; j++)
            {
                        for(k = 0; k < 3; k++)
                        {
                                    decrypt[i][j] = decrypt[i][j] + b[i][k] * encrypt[k][j];
                        }
                        decrypt[i][j] = decrypt[i][j] % 26;
            }
          }
  }
void inverse( int b[][] )
{
   int i, j, k;
   float p, q;
   for(i = 0; i < 3; i++)
      for(j = 0; j < 3; j++)
         {
         if(i == j)
            b[i][j]=1;
         else
            b[i][j]=0;
         }
   for(k = 0; k < 3; k++) {
      for(i = 0; i < 3; i++) {
         p = c[i][k];
         q = c[k][k];
         for(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(i = 0; i < 3; i++){
         for(j = 0; j < 3; j++) {
           b[i][j] = b[i][j] / c[i][i];
         }
     }
}
public static String HillCipher(String message, String key)
{
   int [][]keyMatrix = new int[3][3];
   getKeyMatrix(key, keyMatrix);
   int [][]messageVector = new int[3][1];
   for (int i = 0; i < 3; i++)
```

```
        messageVector[i][0] = (message.charAt(i)) % 65;
    int [][]cipherMatrix = new int[3][1];
    encrypt(cipherMatrix, keyMatrix, messageVector);
    String CipherText="";
    for (int i = 0; i < 3; i++)
        CipherText += (char)(cipherMatrix[i][0] + 65);
    return CipherText;
}
public static void main(String[] args)
{
    final Scanner sc = new Scanner(System.in);
    System.out.println("Enter plaintext");
    final String message = sc.nextLine();
    System.out.println("Enter keyword");
    final String key = sc.nextLine();
    String c=HillCipher(message, key);
    System.out.println("Plain Text   :"+ message);
    System.out.println("Key          :"+ key);
    System.out.println("Cipher Text  :"+ c);
    sc.close();
}
}
```

Output:



```
C:\Users\Admin\Desktop\college\7th Semester\Information and network security (INS)\Lab\Programs>java hillcipher
Enter plaintext
ACT
Enter keyword
GYBNQKURP
Plain Text   :ACT
Key          :GYBNQKURP
Cipher Text  :POH
```

Learning Outcomes:

The basic Hill cipher is vulnerable to plaintext attack because it's completely linear. An opponent who intercepts the plain/ciphertext can easily set up a linear system which can be solved. So finding the solution using a standard linear algebra algorithm takes less time. Also, the key domain is less as they make use of those keys which can be inverted when used in matrix format.

Experiment 6

Aim: Write a program to implement S-DES SubKey Generation algorithm.

Theory: Simplified-DES The S-DES encryption algorithm takes an 8-bit block of plaintext (example: 10111101) and a 10-bit key as input and produces an 8-bit block of ciphertext as output. The S-DES decryption algorithm takes an 8-bit block of ciphertext and the same 10-bit key used to produce that ciphertext as input and produces the original 8-bit block of plaintext.

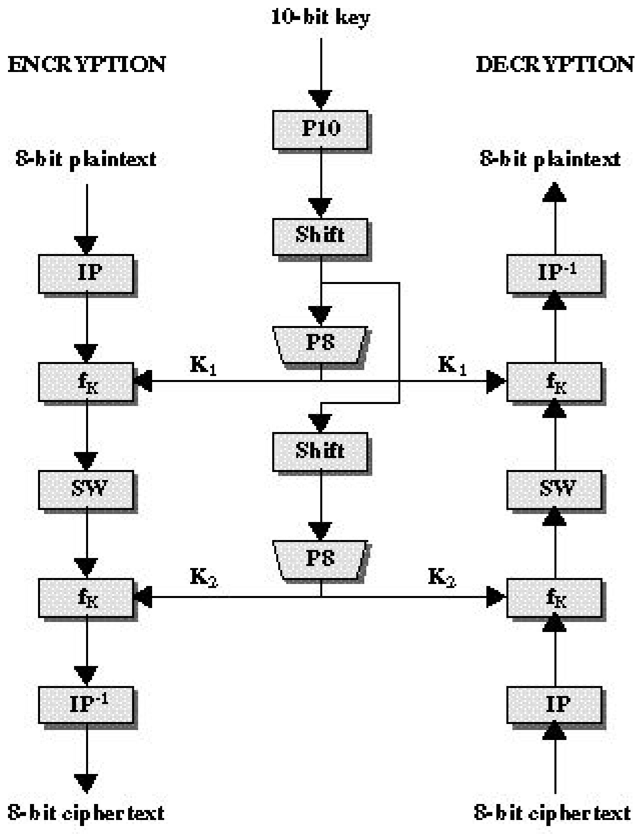


Figure 3.1 Simplified DES Scheme

The encryption algorithm involves five functions:

- an initial permutation (IP)
- a complex function labeled fk, which involves both permutation and substitution operations and depends on a key input
- a simple permutation function that switches (SW) the two halves of the data
- the function fk again
- a permutation function that is the inverse of the initial permutation

The function fk takes as input not only the data passing through the encryption algorithm, but also an 8-bit key. Here a 10-bit key is used from which two 8-bit subkeys are generated. The key is first subjected to a permutation (P10). Then a shift operation is performed. The output of the shift operation then passes through a permutation function that produces an 8-bit output (P8) for the first subkey (K1). The output of the shift operation also feeds into another shift and another instance of P8 to produce the second subkey (K2).

The encryption algorithm can be expressed as a composition of functions, which can be written as where The decryption algorithm can be shown as:

Ciphertext = IP-1 (fK2 (SW (fk1 (IP (plaintext)))))

K1 = P8 (Shift (P10 (Key)))

K2 = P8 (Shift (shift (P10 (Key))))

The decryption algorithm can be shown as:

Plaintext = IP-1 (fK1 (SW (fk2 (IP (ciphertext)))))

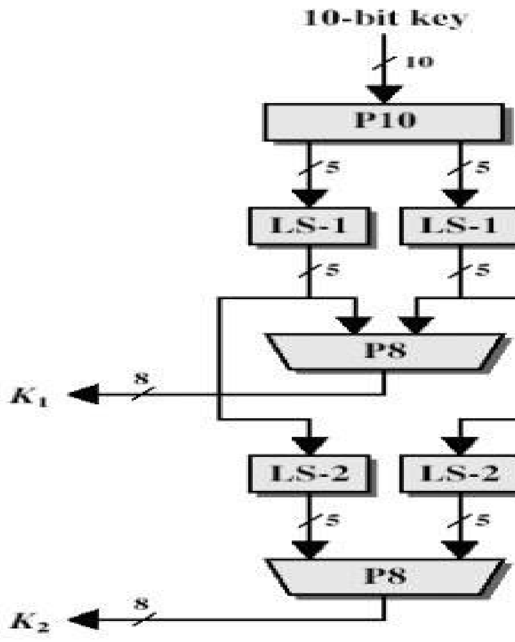


Figure: key generation for S-DES

S-DES key generation

S-DES depends on the use of a 10-bit key shared between sender and receiver. From this key, two 8-bit subkeys are produced for use in particular stages of the encryption and decryption algorithm. First, permute the key in the following fashion. Let the 10-bit key be designated as (k1, k2, k3, k4, k5, k6, k7, k8, k9, k10). Then the permutation P10 is defined as:

P10

3	5	2	7	4	10	1	9	8	6

This table is read from left to right; each position in the table gives the identity of the input bit that produces the output bit in that position. So the first output bit is bit 3 of the input; the second output bit is bit 5 of the input, and so on. For example, the key (1010000010) is permuted to (10000 01100). Next, perform a circular left shift (LS-1), or rotation, separately on the first five bits and the second five bits. In our example, the result is (00001 11000). Next we apply P8, which picks out and permutes 8 of the 10 bits according to the following rule:

P8

6	3	7	4	8	5	10	9

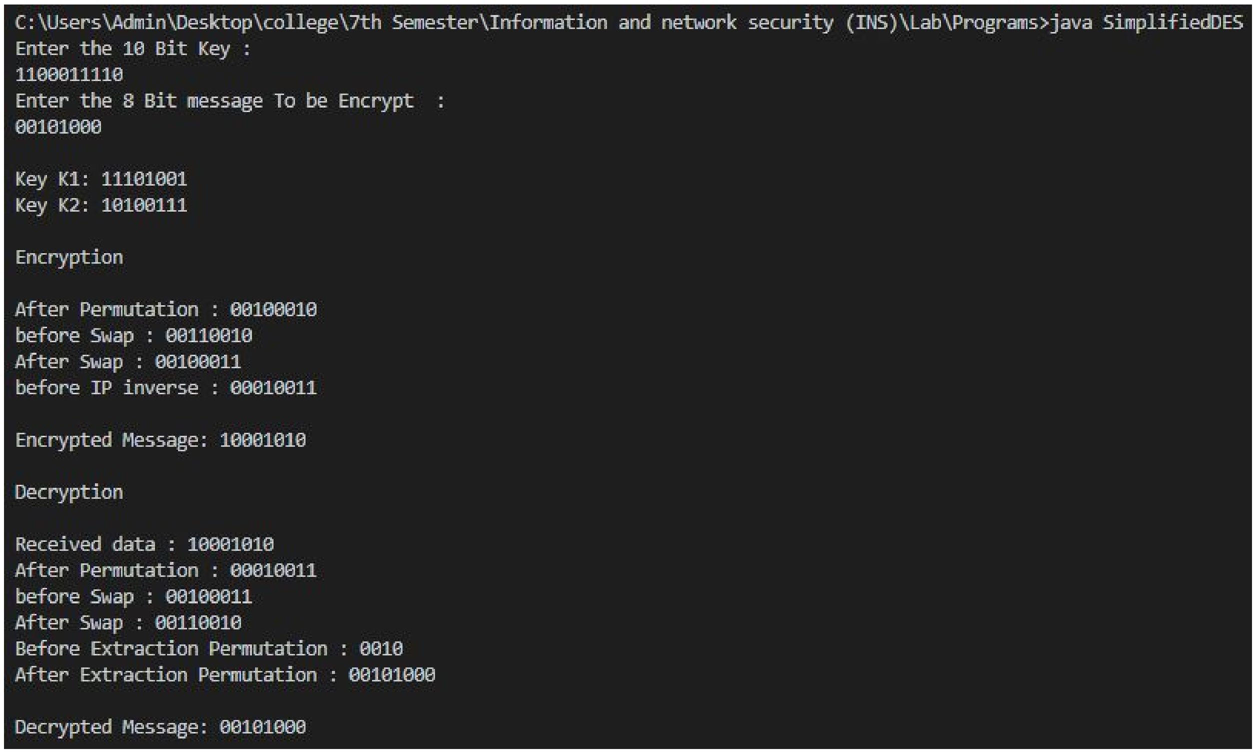
The result is subkey 1 (K1). In our example, this yields (10100100). We then go back to the pair of 5-bit strings produced by the two LS-1 functions and perform a circular left shift of 2 bit positions on each string. In our example, the value (00001 11000) becomes (00100 00011). Finally, P8 is applied again to produce K2. In our example, the result is (01000011).

Source Code:

```
import java.io.*;
import java.lang.*;
class SDES
    {
     public int K1, K2;
     public static final int P10[] = { 3, 5, 2, 7, 4, 10, 1, 9, 8, 6};
     public static final int P10max = 10;
     public static final int P8[] = { 6, 3, 7, 4, 8, 5, 10, 9};
     public static final int P8max = 10;
     public static final int P4[] = { 2, 4, 3, 1};
     public static final int P4max = 4;
     public static final int IP[] = { 2, 6, 3, 1, 4, 8, 5, 7};
     public static final int IPmax = 8;
     public static final int IPI[] = { 4, 1, 3, 5, 7, 2, 8, 6};
     public static final int IPImax = 8;
     public static final int EP[] = { 4, 1, 2, 3, 2, 3, 4, 1};
     public static final int EPmax = 4;
     public static final int S0[][] = {{ 1, 0, 3, 2},{ 3, 2, 1, 0},{ 0, 2, 1,
                                       3},{ 3, 1, 3, 2}};
     public static final int S1[][] = {{ 0, 1, 2, 3},{ 2, 0, 1, 3},{ 3, 0, 1,
                                       2},{ 2, 1, 0, 3}};

    public static int permute( int x, int p[], int pmax)
    {
      int y = 0;
      for( int i = 0; i < p.length; ++i)
       {
         y <<= 1;
         y |= (x >> (pmax - p[i])) & 1;
      }
      return y;
    }

public static int F( int R, int K)
 {
       int t = permute( R, EP, EPmax) ^ K;
       int t0 = (t >> 4) & 0xF;
       int t1 = t & 0xF;
       t0 = S0[ ((t0 & 0x8) >> 2) | (t0 & 1) ][ (t0 >> 1) & 0x3 ];
       t1 = S1[ ((t1 & 0x8) >> 2) | (t1 & 1) ][ (t1 >> 1) & 0x3 ];
        t = permute( (t0 << 2) | t1, P4, P4max);
      return t;
 }

public static int fK( int m, int K)
     {
        int L = (m >> 4) & 0xF;
        int R = m & 0xF;
        return ((L ^ F(R,K)) << 4) | R;
     }

public static int SW( int x)
{
 return ((x & 0xF) << 4) | ((x >> 4) & 0xF);
}

 public byte encrypt( int m)
   {
      System.out.println("\nEncryption");
       m = permute( m, IP, IPmax);
      System.out.print("\nAfter Permutation : ");
       printData( m, 8);
```

```
        m = fK( m, K1);
        System.out.print("\nbefore Swap : ");
        printData( m, 8);
        m = SW( m);
        System.out.print("\nAfter Swap : ");
        printData( m, 8);
        m = fK( m, K2);
        System.out.print("\nbefore IP inverse : ");
        printData( m, 8);
        m = permute( m, IPI, IPImax);
        return (byte) m;
      }

   public byte decrypt( int m)
       {
        System.out.println("\nDecryption");
        System.out.print("\nReceived data : ");
        printData( m, 8);
        m = permute( m, IP, IPmax);
        System.out.print("\nAfter Permutation : ");
        printData( m, 8);
        m = fK( m, K2);
        System.out.print("\nbefore Swap : ");
        printData( m, 8);
        m = SW( m);
        System.out.print("\nAfter Swap : ");
        printData( m, 8);
        m = fK( m, K1);
        System.out.print("\nBefore Extraction Permutation : ");
        printData( m, 4);
        m = permute( m, IPI, IPImax);
        System.out.print("\nAfter Extraction Permutation : ");
        printData( m, 8);
        return (byte) m;
       }

 public static void printData( int x, int n)
        {
         int mask = 1 << (n-1);
        while( mask > 0)
        {
        System.out.print( ((x & mask) == 0) ? '0' : '1');
        mask >>= 1;
        }
       }

   public SDES( int K)
     {
        K = permute( K, P10, P10max);
        int t1 = (K >> 5) & 0x1F;
        int t2 = K & 0x1F;
        t1 = ((t1 & 0xF) << 1) | ((t1 & 0x10) >> 4);
        t2 = ((t2 & 0xF) << 1) | ((t2 & 0x10) >> 4);
        K1 = permute( (t1 << 5)| t2, P8, P8max);
        t1 = ((t1 & 0x7) << 2) | ((t1 & 0x18) >> 3);
        t2 = ((t2 & 0x7) << 2) | ((t2 & 0x18) >> 3);
        K2 = permute( (t1 << 5)| t2, P8, P8max);
     }
     }

public class SimplifiedDES
     {
       public static void main( String args[]) throws Exception
       {
        DataInputStream inp = new DataInputStream(System.in);
        System.out.println("Enter the 10 Bit Key :");
        int K = Integer.parseInt(inp.readLine(),2);
        SDES A = new SDES( K);
        System.out.println("Enter the 8 Bit message To be Encrypt  : ");
        int m = Integer.parseInt(inp.readLine(),2);
        System.out.print("\nKey K1: ");
        SDES.printData( A.K1, 8);
        System.out.print("\nKey K2: ");
        SDES.printData( A.K2, 8);
        System.out.print("\n");

        m = A.encrypt( m);
        System.out.print("\n\nEncrypted Message: ");
        SDES.printData( m, 8);
        System.out.print("\n");
        m = A.decrypt( m);
        System.out.print("\n");
        System.out.print("\nDecrypted Message: ");
        SDES.printData( m, 8);
        System.out.print("\n");
      }
   }
```

Output:



```
C:\Users\Admin\Desktop\college\7th Semester\Information and network security (INS)\Lab\Programs>java SimplifiedDES
Enter the 10 Bit Key :
1100011110
Enter the 8 Bit message To be Encrypt  :
00101000

Key K1: 11101001
Key K2: 10100111

Encryption

After Permutation : 00100010
before Swap : 00110010
After Swap : 00100011
before IP inverse : 00010011

Encrypted Message: 10001010

Decryption

Received data : 10001010
After Permutation : 00010011
before Swap : 00100011
After Swap : 00110010
Before Extraction Permutation : 0010
After Extraction Permutation : 00101000

Decrypted Message: 00101000
```

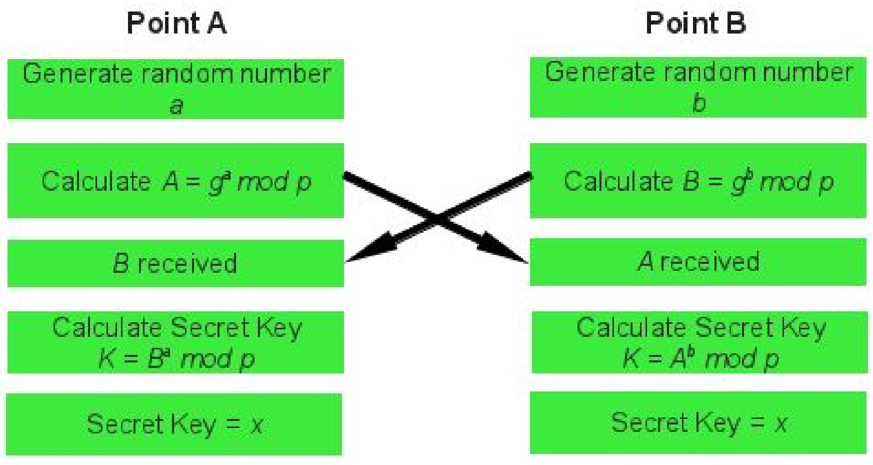
Learning Outcomes:

DES ciphers are hard to break using the brute force approach. Although number of permutations is not too much, the use of 2 or 3 keys in DES makes it difficult to crack. Using differential cryptanalysis can however prove to be more efficient than brute force attack in DES , still from calculations and observations we see that DES is resistant to cryptanalysis.

Experiment 7

Aim: Write a program to implement key exchange using Diffie-Hallman key exchange.

Theory: Diffie-Hellman key exchange, also called exponential key exchange, is a method of digital encryption that uses numbers raised to specific powers to produce decryption keys on the basis of components that are never directly transmitted, making the task of a would-be code breaker mathematically overwhelming. To implement Diffie-Hellman, the two end users Alice and Bob, while communicating over a channel they know to be private, mutually agree on positive whole numbers p and q, such that p is a prime number and q is a generator of p. The generator q is a number that, when raised to positive whole-number powers less than p, never produces the same result for any two such whole numbers.


Point A
Point B
Generate random number a
Generate random number b
Calculate $A = g^a \bmod p$
Calculate $B = g^b \bmod p$
B received
A received
Calculate Secret Key $K = B^a \bmod p$
Calculate Secret Key $K = A^b \bmod p$
Secret Key = x
Secret Key = x

p = prime number, g = generator

The value of p may be large but the value of q is usually small. Once Alice and Bob have agreed on p and q in private, they choose positive whole-number personal keys a and b, both less than the prime-number modulus p. Neither user divulges their personal key to anyone; ideally they memorize these numbers and do not write them down or store them anywhere. Next, Alice and Bob compute public keys a* and b* based on their personal keys according to the formulas: -

a* = [q^a] mod p

b* = [q^b] mod p

The two users can share their public keys a* and b* over a communications medium assumed to be insecure, such as the Internet or a corporate wide area network (WAN). From these public keys, x and y can be generated by either user on the basis of their own personal keys.

Alice computes x using the formula

x = [(b*)^a] mod p

Bob computes x using the formula

y = [(a*)^b] mod p

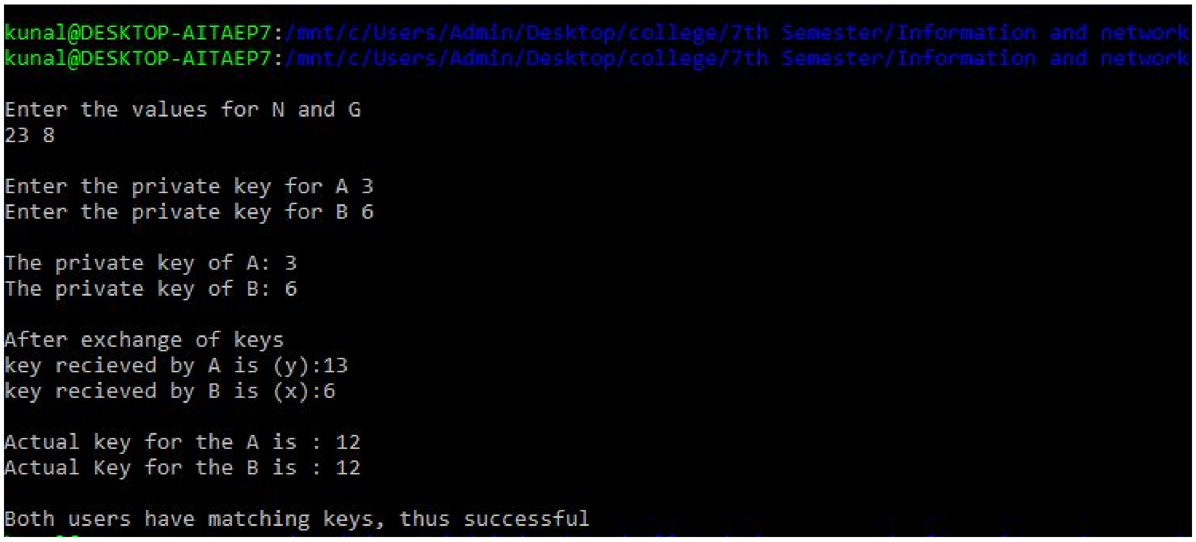
The value of x and y turns out to be the same according to either of the above two formulas. The two users can therefore, in theory, communicate privately over a public medium with an encryption method of their choice using the decryption key x.

Source Code:

```
#include <iostream>
#include<math.h>
using namespace std;
long long int calc(long long int a, long long int b, long long int N)
{
   if (b == 1)
      return a;
   else
      return (((long long int)pow(a, b)) % N);
}

int main() {
   long long int N, G, x, a, y, b, ka, kb;
   cout<<"\nEnter the values for N and G\n";
   cin>>N>>G;
   cout<<"\nEnter the private key for A ";
   cin>>a;
   cout<<"Enter the private key for B ";
   cin>>b;
   cout<<"\nThe private key of A: "<<a;
   cout<<"\nThe private key of B: "<<b;

   x = calc(G, a, N);
   y = calc(G, b, N);
   cout<<"\n\nAfter exchange of keys";
   cout<<"\nkey recieved by A is (y):"<<y;
   cout<<"\nkey recieved by B is (x):"<<x;
```

```
    ka = calc(y, a, N);
    kb = calc(x, b, N);
    cout<<"\n\nActual key for the A is : "<<ka;
    cout<<"\nActual Key for the B is : "<<kb;
    if(ka= =kb)
        cout<<"\n\nBoth users have matching keys, thus successful";
    else
        cout<<"Key Generation failed";
    return 0;
}
```

Output:



```
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/college/7th Semester/Information and network
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/college/7th Semester/Information and network

Enter the values for N and G
23 8

Enter the private key for A 3
Enter the private key for B 6

The private key of A: 3
The private key of B: 6

After exchange of keys
key recieved by A is (y):13
key recieved by B is (x):6

Actual key for the A is : 12
Actual Key for the B is : 12

Both users have matching keys, thus successful
```

Learning Outcomes:

The personal keys a and b, which are critical in the calculation of x, have not been transmitted over a public medium. Because it is a large and apparently random number, a potential hacker has almost no chance of correctly guessing x, even with the help of a powerful computer to conduct millions of trials.

<u>Experiment 8</u>

Aim: Write a program to implement encryption and decryption using the RSA algorithm.

Theory: RSA is a cryptosystem for public-key encryption, and is widely used for securing sensitive data, particularly when being sent over an insecure network such as the Internet. Public-key cryptography, also known as asymmetric cryptography, uses two different but mathematically linked keys, one public and one private. The public key can be shared with everyone, whereas the private key must be kept secret.
Many protocols like SSH, OpenPGP, S/MIME, and SSL/TLS rely on RSA for encryption and digital signature functions. It is also used in software programs -- browsers are an obvious example, which need to establish a secure connection over an insecure network like the Internet or validate a digital signature. RSA signature verification is one of the most commonly performed operations in IT.
Explaining RSA's popularity RSA derives its security from the difficulty of factoring large integers that are the product of two large prime numbers. Multiplying these two numbers is easy, but determining the original prime numbers from the total -- factoring -- is considered infeasible due to the time it would take even using today's supercomputers.

The public and the private key-generation algorithm is the most complex part of RSA cryptography. Two large prime numbers, p and q, are generated using the Rabin-Miller primality test algorithm. A modulus n is calculated by multiplying p and q. This number is used by both the public and private keys and provides the link between them. Its length, usually expressed in bits, is called the key length. The public key consists of the modulus n, and a public exponent, e, which is normally set at 65537, as it's a prime number that is not too large. The e figure doesn't have to be a secretly selected prime number as the public key is shared with everyone. The private key consists of the modulus n and the private exponent d, which is calculated using the Extended Euclidean algorithm to find the multiplicative inverse with respect to the totient of n.

- <u>Encryption</u>
  Sender A does the following: -
    1. Obtains the recipient B's public key (n,e).
    2. Represents the plaintext message as a positive integer m with 1<m<n.
    3. Computes the ciphertext c = (me mod n) and Sends the ciphertext c to B.
- <u>Decryption</u>
  Recipient B does the following: -
    1. Uses his private key (n,d) to compute m=cd mod n.
    2. Extracts the plaintext from the message representative m.

Source Code:

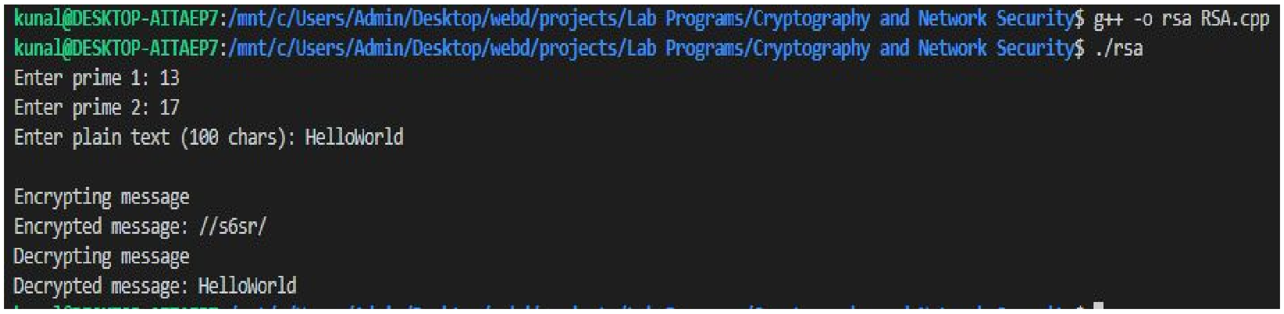
```
#include<iostream>
#include<cstdio>
#include<math.h>
#include<string.h>
#include<stdlib.h>
using namespace std;
int p, q, n, t, flag, e[100], d[100], temp[100], j, m[100],
en[100], i;
char msg[100];
int prime(long int pr)
        {
        int i;
        j = sqrt(pr);
        for (i = 2; i <= j; i++)
        {if (pr % i == 0)
        return 0;}
        return 1;
        }
void ce()
        {
        int k;
        k = 0;
        for (i = 2; i < t; i++)
        {if (t % i == 0)
        continue;
        flag = prime(i);
        if (flag == 1 && i != p && i != q)
        {e[k] = i;
        flag = cd(e[k]);
        if (flag > 0)
        {d[k] = flag;
          k++;}
        if (k == 99)
        break;
        }}
        }
```

```
          long int cd(long int x)
          {long int k = 1;
          while (1)
          {
          k = k + t;
          if (k % x == 0)
          return (k / x);}
          }
void encrypt()
          {
          long int pt, ct, key = e[0], k, len;
          i = 0;
          len = strlen(msg);
          while (i != len)
          {pt = m[i];
          pt = pt - 96;
          k = 1;
          for (j = 0; j < key; j++)
          {k = k * pt;
          k = k % n;
          }
          temp[i] = k;
          ct = k + 96;
          en[i] = ct;
          i++;
          }
          en[i] = -1;
          cout << "Encrypted message: ";
          for (i = 0; en[i] != -1; i++)
          printf("%c", en[i]);
          }
void decrypt()
          {
          long int pt, ct, key = d[0], k;
          i = 0;
          while (en[i] != -1)
          {ct = temp[i];
          k = 1;
          for (j = 0; j < key; j++)
```

```
        {k = k * ct;
        k = k % n;}
        pt = k + 96;
        m[i] = pt;
        i++;
        }
        m[i] = -1;
        cout << "Decrypted message: ";
        for (i = 0; m[i] != -1; i++)
        printf("%c", m[i]);
        }
int main()
        {
        cout << "Enter prime 1: ";
        cin >> p;
        flag = prime(p);
        if (flag == 0)
        {cout << "Entered Value is not PRIME Exiting";
        exit(1);}
        cout << "Enter prime 2: ";
        cin >> q;
        flag = prime(q);
        if (flag == 0 || p == q)
        {cout << "Entered Value is not PRIME Exiting";
        exit(1);}
        cout << "Enter plain text (100 chars): ";
        fflush(stdin);
        cin >> msg;
        for (i = 0; msg[i] != '\0'; i++)
        m[i] = msg[i];
        n = p * q;
        t = (p - 1) * (q - 1);
        ce();
        cout << endl << "Encrypting message" << endl;
        encrypt();
        cout << endl << "Decrypting message" << endl;
        decrypt();
        cout << endl;
        return 0;
```

}

Output:



```
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/webd/projects/Lab Programs/Cryptography and Network Security$ g++ -o rsa RSA.cpp
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/webd/projects/Lab Programs/Cryptography and Network Security$ ./rsa
Enter prime 1: 13
Enter prime 2: 17
Enter plain text (100 chars): HelloWorld

Encrypting message
Encrypted message: //s6sr/
Decrypting message
Decrypted message: HelloWorld
```

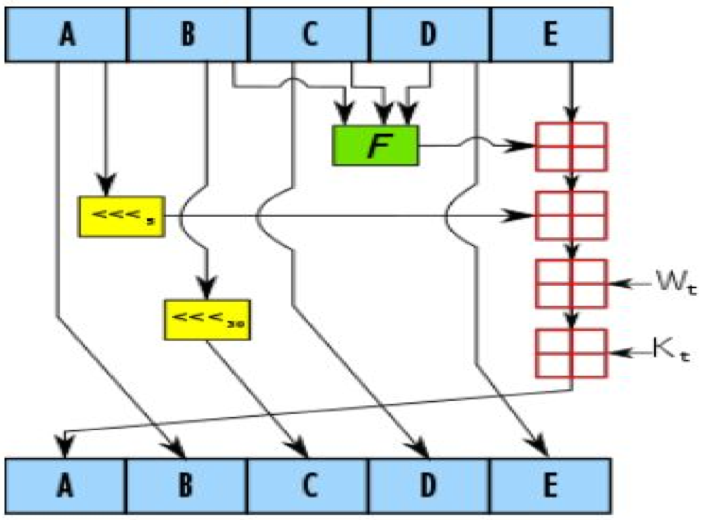
Learning Outcomes:

In RSA cryptography, both the public and the private keys can encrypt a message; the opposite key from the one used to encrypt a message is used to decrypt it. This attribute is one reason why RSA has become the most widely used asymmetric algorithm.

Experiment 9

Aim: Write a program to implement generation of hash by SHA1.

Theory: In computer cryptography, a popular message compress standard is utilized known as Secure Hash Algorithm (SHA). Its enhanced version is called SHA-1. It has the ability to compress a fairly lengthy message and create a short message abstract in response. The algorithm can be utilized along various protocols to ensure security of the applied algorithm, particularly for Digital Signature Standard (DSS). The algorithm offers five separate hash functions which were created by the National Security Agency (NSA) and were issued by the National Institute of Standards and Technology (NIST).



According to SHA-1 standard, a message digest is evaluated utilizing padded messages. The evaluation utilizes two buffers, each comprising five 32 bit words and a sequence of eighty 32 bit words. The words of the first five-word buffer are labelled as A, B, C, D and E. The words of the second five-word buffer are labelled as H0, H1, H2, H3 and H4. The words of the eighty-word sequence are labelled as W0, W1, W2 to W79. SHA1 operates blocks of 512 bits, when evaluating a message digest. The entire extent lengthwise of message digest shall be multiple of 512. A novel architecture of SHA-1 for enhanced throughput and decreased area, in which at the same time diverse acceleration techniques are exerted like pre-computation, loop unfolding and pipelining. Hash function requires a set of operations that are an input of diversifying length and creating a stable length string which is known as the hash value or message digest.

Pre-computation technique is utilized to produce definite intermediate signals of the critical path and reserve them in a register, which can be utilized in the computation of values of the next step. For a message possessing a maximum length of 264, SHA-1 constructs a 160 bit message digest. ` 160 bit dedicated hash function is incorporated in SHA-1 originate in the design principle of MD4, which is an algorithm utilized to certify data integrity through the formation of a 128 bit message digest from data input that is declared to be as distinctive to that particular data as a fingerprint is to the particular individual.

The input message is padded and broken into 'k' 512 bit message blocks. At every iteration of the compression function 'h', a 160 bit chaining variable Ht is upgraded utilizing one message block Mt+1, that is **Ht+1 = h(Ht, Mt+1).**

The beginning value H0 is established in advance and Hk is the out-turn of the hash function. SHA-1 compression function is constructed upon the Davis Meyer construction. It utilizes a function 'E' as a block cipher with Ht for the message input and Mt+1 for the key input. The SHA-1 is implicit. It is as secure as anything in opposition to reimaged attacks, although it is effortless to calculate, which means it is uncomplicated to mount a brute force or dictionary attack. It is a well-known cryptographic primitive which ensures the integrity and reliability of original messages.

Source Code:

sha1.h

```
#ifndef SHA1_HPP
#define SHA1_HPP
#include <cstdint>
#include <iostream>
#include <string>
#include <stdint.h>
class SHA1
{
public:
 SHA1();
 void update(const std::string &s);
 void update(std::istream &is);
 std::string final();
 static std::string from_file(const std::string &filename);
private:
 uint32_t digest[5];
 std::string buffer;
 uint64_t transforms;
};
#endif
```

SHA.cpp

```
#include "sha1.h"
#include <sstream>
#include <iomanip>
#include <fstream>
static const size_t BLOCK_INTS = 16;
static const size_t BLOCK_BYTES = BLOCK_INTS * 4;
static void reset(uint32_t digest[], std::string &buffer, uint64_t &transforms)
{
 digest[0] = 0x67452301;
 digest[1] = 0xefcdab89;
 digest[2] = 0x98badcfe;
 digest[3] = 0x10325476;
 digest[4] = 0xc3d2e1f0;
  buffer = "";
 transforms = 0;
}
static uint32_t rol(const uint32_t value, const size_t bits)
{ return (value << bits) | (value >> (32 - bits));}
static uint32_t blk(const uint32_t block[BLOCK_INTS], const size_t i)
{  return rol(block[(i+13)&15] ^ block[(i+8)&15] ^ block[(i+2)&15] ^ block[i], 1); }

static void R0(const uint32_t block[BLOCK_INTS], const uint32_t v, uint32_t &w, const
uint32_t x, const uint32_t y, uint32_t &z, const size_t i)
{
 z += ((w&(x^y))^y) + block[i] + 0x5a827999 + rol(v, 5);
 w = rol(w, 30);
}
static void R1(uint32_t block[BLOCK_INTS], const uint32_t v, uint32_t &w, const uint32_t x,
const uint32_t y, uint32_t &z, const size_t i)
{
 block[i] = blk(block, i);
 z += ((w&(x^y))^y) + block[i] + 0x5a827999 + rol(v, 5);
 w = rol(w, 30);
}
static void R2(uint32_t block[BLOCK_INTS], const uint32_t v, uint32_t &w, const uint32_t x,
const uint32_t y, uint32_t &z, const size_t i)
```

```
{
 block[i] = blk(block, i);
 z += (w^x^y) + block[i] + 0x6ed9eba1 + rol(v, 5);
w = rol(w, 30);
}
static  void R3(uint32_t block[BLOCK_INTS], const uint32_t v, uint32_t &w, const uint32_t x,
const uint32_t y, uint32_t &z, const size_t i)
{
 block[i] = blk(block, i);
 z += (((w|x)&y)|(w&x)) + block[i] + 0x8f1bbcdc + rol(v, 5);
 w = rol(w, 30);
}
static  void R4(uint32_t block[BLOCK_INTS], const uint32_t v, uint32_t &w, const uint32_t x,
const uint32_t y, uint32_t &z, const size_t i)
{
 block[i] = blk(block, i);
 z += (w^x^y) + block[i] + 0xca62c1d6 + rol(v, 5);
 w = rol(w, 30);
}
static void transform(uint32_t digest[], uint32_t block[BLOCK_INTS], uint64_t &transforms)
{
 uint32_t a = digest[0];
 uint32_t b = digest[1];
 uint32_t c = digest[2];
 uint32_t d = digest[3];
 uint32_t e = digest[4];
 R0(block, a, b, c, d, e, 0);
 R0(block, e, a, b, c, d, 1);
 R0(block, d, e, a, b, c, 2);
 R0(block, c, d, e, a, b, 3);
 R0(block, b, c, d, e, a, 4);
 R0(block, a, b, c, d, e, 5);
 R0(block, e, a, b, c, d, 6);
 R0(block, d, e, a, b, c, 7);
 R0(block, c, d, e, a, b, 8);
 R0(block, b, c, d, e, a, 9);
 R0(block, a, b, c, d, e, 10);
 R0(block, e, a, b, c, d, 11);
 R0(block, d, e, a, b, c, 12);
```

```
R0(block, c, d, e, a, b, 13);
R0(block, b, c, d, e, a, 14);
R0(block, a, b, c, d, e, 15);
R1(block, e, a, b, c, d, 0);
R1(block, d, e, a, b, c, 1);
R1(block, c, d, e, a, b, 2);
R1(block, b, c, d, e, a, 3);
R2(block, a, b, c, d, e, 4);
R2(block, e, a, b, c, d, 5);
R2(block, d, e, a, b, c, 6);
R2(block, c, d, e, a, b, 7);
R2(block, b, c, d, e, a, 8);
R2(block, a, b, c, d, e, 9);
R2(block, e, a, b, c, d, 10);
R2(block, d, e, a, b, c, 11);
R2(block, c, d, e, a, b, 12);
R2(block, b, c, d, e, a, 13);
R2(block, a, b, c, d, e, 14);
R2(block, e, a, b, c, d, 15);
R2(block, d, e, a, b, c, 0);
R2(block, c, d, e, a, b, 1);
R2(block, b, c, d, e, a, 2);
R2(block, a, b, c, d, e, 3);
R2(block, e, a, b, c, d, 4);
R2(block, d, e, a, b, c, 5);
R2(block, c, d, e, a, b, 6);
R2(block, b, c, d, e, a, 7);
R3(block, a, b, c, d, e, 8);
R3(block, e, a, b, c, d, 9);
R3(block, d, e, a, b, c, 10);
R3(block, c, d, e, a, b, 11);
R3(block, b, c, d, e, a, 12);
R3(block, a, b, c, d, e, 13);
R3(block, e, a, b, c, d, 14);
R3(block, d, e, a, b, c, 15);
R3(block, c, d, e, a, b, 0);
R3(block, b, c, d, e, a, 1);
R3(block, a, b, c, d, e, 2);
R3(block, e, a, b, c, d, 3);
```

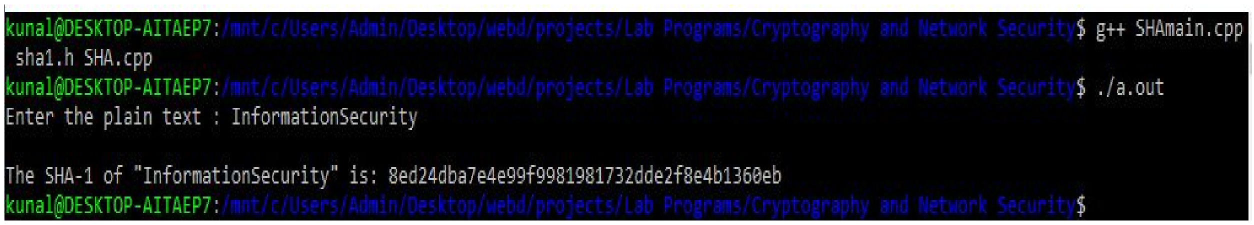
```
 R3(block, d, e, a, b, c, 4);
 R3(block, c, d, e, a, b, 5);
 R3(block, b, c, d, e, a, 6);
 R3(block, a, b, c, d, e, 7);
 R3(block, e, a, b, c, d, 8);
 R3(block, d, e, a, b, c, 9);
 R3(block, c, d, e, a, b, 10);
 R3(block, b, c, d, e, a, 11);
 R4(block, a, b, c, d, e, 12);
 R4(block, e, a, b, c, d, 13);
 R4(block, d, e, a, b, c, 14);
 R4(block, c, d, e, a, b, 15);
 R4(block, b, c, d, e, a, 0);
 R4(block, a, b, c, d, e, 1);
 R4(block, e, a, b, c, d, 2);
 R4(block, d, e, a, b, c, 3);
 R4(block, c, d, e, a, b, 4);
 R4(block, b, c, d, e, a, 5);
 R4(block, a, b, c, d, e, 6);
 R4(block, e, a, b, c, d, 7);
 R4(block, d, e, a, b, c, 8);
 R4(block, c, d, e, a, b, 9);
 R4(block, b, c, d, e, a, 10);
 R4(block, a, b, c, d, e, 11);
 R4(block, e, a, b, c, d, 12);
 R4(block, d, e, a, b, c, 13);
 R4(block, c, d, e, a, b, 14);
 R4(block, b, c, d, e, a, 15);
 digest[0] += a;
 digest[1] += b;
 digest[2] += c;
 digest[3] += d;
 digest[4] += e;
 transforms++;
}
static void buffer_to_block(const std::string &buffer, uint32_t block[BLOCK_INTS])
{
 for (size_t i = 0; i < BLOCK_INTS; i++)
 {
```

```
 block[i] = (buffer[4*i+3] & 0xff)
 | (buffer[4*i+2] & 0xff)<<8
 | (buffer[4*i+1] & 0xff)<<16
 | (buffer[4*i+0] & 0xff)<<24;
 }
}
SHA1::SHA1()
{
 reset(digest, buffer, transforms);
}
void SHA1::update(const std::string &s)
{
 std::istringstream is(s);
 update(is);
}
void SHA1::update(std::istream &is)
{
 while (true)
 {
 char sbuf[BLOCK_BYTES];
 is.read(sbuf, BLOCK_BYTES - buffer.size());
 buffer.append(sbuf, (std::size_t)is.gcount());
 if (buffer.size() != BLOCK_BYTES)
 {return;}
 uint32_t block[BLOCK_INTS];
 buffer_to_block(buffer, block);
 transform(digest, block, transforms);
 buffer.clear();
 }
}
std::string SHA1::final()
{
uint64_t total_bits = (transforms*BLOCK_BYTES + buffer.size()) * 8;

 buffer += (char)0x80;
 size_t orig_size = buffer.size();
 while (buffer.size() < BLOCK_BYTES)
 {buffer += (char)0x00;}
 uint32_t block[BLOCK_INTS];
 buffer_to_block(buffer, block);
 if (orig_size > BLOCK_BYTES - 8)
 {
 transform(digest, block, transforms);
 for (size_t i = 0; i < BLOCK_INTS - 2; i++)
 { block[i] = 0;}
}
 block[BLOCK_INTS - 1] = (uint32_t)total_bits;
 block[BLOCK_INTS - 2] = (uint32_t)(total_bits >> 32);
 transform(digest, block, transforms);
 std::ostringstream result;
 for (size_t i = 0; i < sizeof(digest) / sizeof(digest[0]); i++)
 {result << std::hex << std::setfill('0') << std::setw(8);
 result << digest[i];
 }
 reset(digest, buffer, transforms);
 return result.str();
}
std::string SHA1::from_file(const std::string &filename)
{
 std::ifstream stream(filename.c_str(), std::ios::binary);
 SHA1 checksum;
 checksum.update(stream);
 return checksum.final();
}
```

SHAmain.cpp

```
#include "sha1.h"
#include <string>
#include <iostream>
using namespace std;
int main(int /* argc */, const char ** /* argv */)
{cout<<"Enter the plain text : ";
 string input;
 cin>>input;
 SHA1 checksum;
 checksum.update(input);
 const string hash = checksum.final();
 cout << "\nThe SHA-1 of \"" << input << "\" is: " << hash << endl;
 return 0;
}
```

Output:



```
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/webd/projects/Lab Programs/Cryptography and Network Security$ g++ SHAmain.cpp
 sha1.h SHA.cpp
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/webd/projects/Lab Programs/Cryptography and Network Security$ ./a.out
Enter the plain text : InformationSecurity

The SHA-1 of "InformationSecurity" is: 8ed24dba7e4e99f9981981732dde2f8e4b1360eb
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/webd/projects/Lab Programs/Cryptography and Network Security$
```

Learning Outcomes:

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.

To calculate cryptographic hashing value in Java, MessageDigest Class is used, under the package java.security.MessagDigest Class provides the following cryptographic hash function to find the hash value of a text.

Experiment 10

Aim: Write a program to implement the DSA algorithm and verify it in DSS.

Theory: The Digital Signature Algorithm (DSA) is a Federal Information Processing Standard for digital signatures, based on the mathematical concept of modular exponentiations and the discrete logarithm problem. The DSA algorithm works in the framework of public-key cryptosystems and is based on the algebraic properties of the modular exponentiations, together with the discrete logarithm problem (which is considered to be computationally intractable). Messages are signed by the signer's private key and the signatures are verified by the signer's corresponding public key. The digital signature provides message authentication, integrity and non-repudiation.

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

- Choose a prime number q, which is called the prime divisor.
- Choose another prime number p, such that p-1 mod q = 0. p is called the prime modulus.
- Choose an integer g, such that 1 < g < p, g**q mod p = 1 and g = h**((p–1)/q) mod p. q is also called g's multiplicative order modulo p.
- Choose an integer, such that 0 < x < q.
- Compute y as g**x mod p.
- Package the public key as {p,q,g,y}.
- Package the private key as {p,q,g,x}.

The second part of the DSA algorithm is the signature generation and signature verification, which can be described as:

To generate a message signature, the sender can follow these steps:

- Generate the message digest 'h' and a random number 'k', such that 0 < k < q.
- Compute r as (g**k mod p) mod q. If r = 0, select a different k.
- Compute i, such that k*i mod q = 1. it is called the modular multiplicative inverse of k modulo q.
- Compute s = i*(h+r*x) mod q. If s = 0, select a different k.
- Package the digital signature as {r,s}.

To verify a message signature, the receiver of the message and the digital signature can follow these steps:

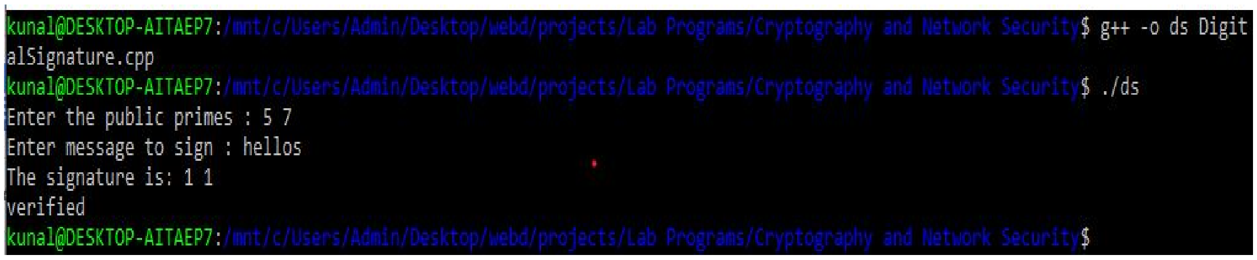
- Generate the message digest h, using the same hash algorithm.
- Compute w, such that s*w mod q = 1. w is called the modular multiplicative inverse of s modulo q.
- Compute u1 = h*w mod q and Compute u2 = r*w mod q.
- Compute v = (((g**u1)*(y**u2)) mod p) mod q.

- If v == r, the digital signature is valid.

Source Code:

```
#include<bits/stdc++.h>
using namespace std;
long long int power(long long int x, long long int y, long long int p)
{
 long long int res = 1; // Initialize result

 x = x % p; // Update x if it is more than or
 // equal to p

 while (y > 0)
 {
 // If y is odd, multiply x with result
 if (y & 1)
 res = (res*x) % p;

 // y must be even now
 y = y>>1; // y = y/2
 x = (x*x) % p;
 }
 return res;
}
long long Hash(string S, int p)
{
 long long hash = 1;
 int prev = 1;
 for (int i = 0; i < S.length(); i++)
 {
 hash = (hash + int(S[i])*prev % p);
 prev = int(S[i]);
 }
 return hash;
}

long long int modInverse(long long int a, long long int m)
{
 long long int m0 = m;
 long long int y = 0, x = 1;
 if (m == 1)
 return 0;

 while (a > 1)
 {
 // q is quotient
 long long int q = a / m;
 long long int t = m;

 // m is remainder now, process same as
 // Euclid's algo
 m = a % m, a = t;
 t = y;

 // Update y and x
 y = x - q * y;
 x = t;
 }

 // Make x positive
 if (x < 0)
 x += m0;

 return x;
}
class DSA
{
private:
 long long int x,g,y,p,q;

public:
 DSA(int , int );
 pair<long long int, long long int> sign(string);
 bool verify(string, pair<long long int, long long int>);
};

DSA::DSA(int P, int Q)
{
 x = rand() % 100 + 1;
 p = P;
 q = Q;
 g = power(2, (p-1)/q, p);
 y = power(g, x, p);
}
pair<long long int, long long int> DSA::sign(string s)
{
 long long int r,s1 = 0,s2 = 0;
 do
  {
 r = rand() % q + 1;
 s1 = power(g, r, p) % q;
 long long k = modInverse(r, q);
 s2 = (k * (Hash(s, p) + x * s1)) % q;
 }while(s1 == 0 || s2 == 0);
 return make_pair(s1, s2);
}
bool DSA::verify(string s, pair<long long int, long long int> sign)
{
 long long int h = Hash(s, p);
 long long int w, u1, u2, v;
 w = modInverse(sign.second, p);
 u1 = (h * w);
 // cout << u1 << endl;
 u2 = (sign.first * w);
 // cout << u2 << endl;
 v = ((power(g, u1, p) * power(y, u2, p))%p)%q;
 // cout << v << endl;
 return (abs(v) == sign.first);
}
int main()
{
 long long int p, q;
 cout << "Enter the public primes : ";
 cin >> p >> q;
 cout << "Enter message to sign : ";
 string s;
 cin >> s;
 DSA D(p, q);
 pair<long long int, long long int> sin = D.sign(s);
 cout << "The signature is: " << sin.first << ' ' << sin.second<< endl;
 if (D.verify(s, sin))
 cout << "verified" << endl;
 else
 cout <<"Rejected" <<endl;
}
```

Output:



```
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/webd/projects/Lab Programs/Cryptography and Network Security$ g++ -o ds Digit
alSignature.cpp
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/webd/projects/Lab Programs/Cryptography and Network Security$ ./ds
Enter the public primes : 5 7
Enter message to sign : hellos
The signature is: 1 1
verified
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/webd/projects/Lab Programs/Cryptography and Network Security$
```

Learning Outcomes:

Digital signature is used to verify authenticity, integrity, non-repudiation ,i.e. it is assuring that the message is sent by the known user and not modified. Thus, digital signatures are used for security. Most websites use digital certificates to enhance trust of their users.

Experiment 11

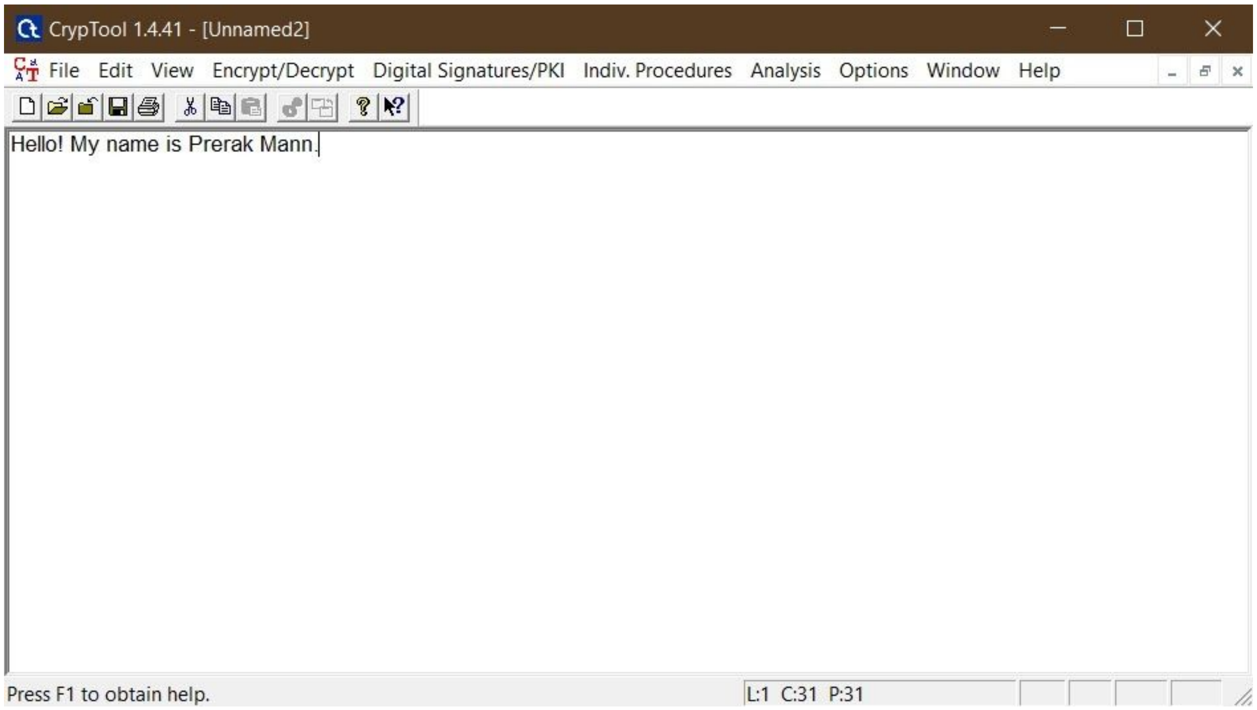
Aim: Perform various encryption-decryption techniques with cryptool.

Theory: CrypTool is an open-source windows program that focuses on the free e-learning software CrypTool illustrating cryptographic and cryptanalytic concepts. According to "Hakin9", CrypTool is worldwide the most widespread e-learning software in the field of cryptology.

CrypTool implements more than 400 algorithms. Users can adjust these with their own parameters. To introduce users to the field of cryptography, the organization created multiple graphical interface software containing an online documentation, analytic tools and algorithms. They contain most classical ciphers, as well as modern symmetric and asymmetric cryptography including RSA, ECC, digital signatures, hybrid encryption, homomorphic encryption, and Diffie–Hellman key exchange. Methods from the area of quantum cryptography (like BB84 key exchange protocol) and the area of post-quantum cryptography (like McEliece, WOTS, Merkle-Signature-Scheme, XMSS, XMSS_MT, and SPHINCS) are implemented. In addition to the algorithms, solvers (analyzers) are included, especially for classical ciphers. Other methods (for instance Huffman code, AES, Keccak, MSS) are visualized.

Output:

Plain Text


CrypTool 1.4.41 - [Unnamed2]
File Edit View Encrypt/Decrypt Digital Signatures/PKI Indiv. Procedures Analysis Options Window Help

Hello! My name is Prerak Mann.

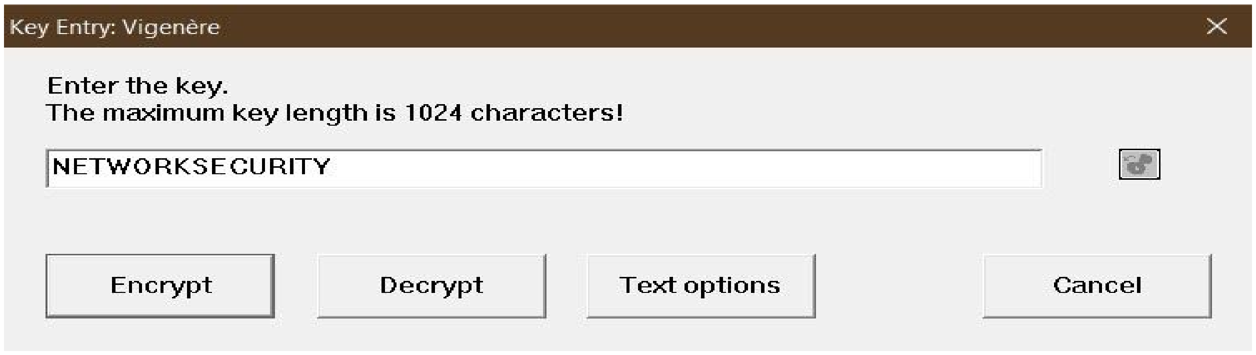
Press F1 to obtain help.

L:1 C:31 P:31

Vigenere Cipher

To facilitate encryption, all the alphabets are usually written out in a large table, traditionally called a tableau. Usually the tableau is 26 × 26, so that 26 full ciphertext alphabets are available. The method of filling the tableau, and of choosing which alphabet to use next, defines the particular polyalphabetic cipher. All such ciphers are easier to break than were believed since the substitution alphabets are repeated for sufficiently large plaintexts.

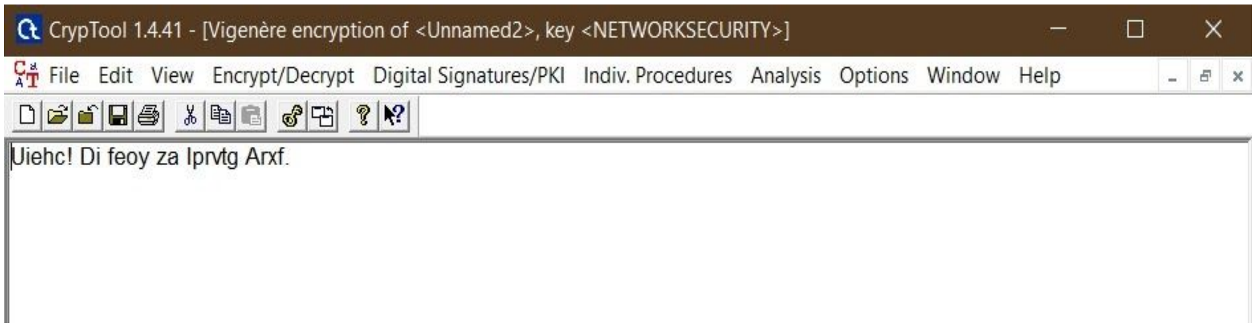
Key


Key Entry: Vigenère

Enter the key.
The maximum key length is 1024 characters!

NETWORKSECURITY

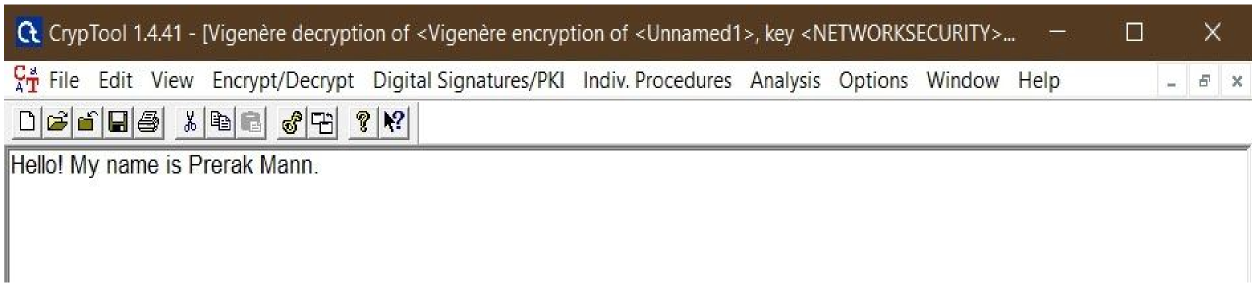
Encryption


CrypTool 1.4.41 - [Vigenère encryption of <Unnamed2>, key <NETWORKSECURITY>]

File Edit View Encrypt/Decrypt Digital Signatures/PKI Indiv. Procedures Analysis Options Window Help

Uiehc! Di feoy za Iprvtg Arxf.

Decryption


CrypTool 1.4.41 - [Vigenère decryption of <Vigenère encryption of <Unnamed1>, key <NETWORKSECURITY>...

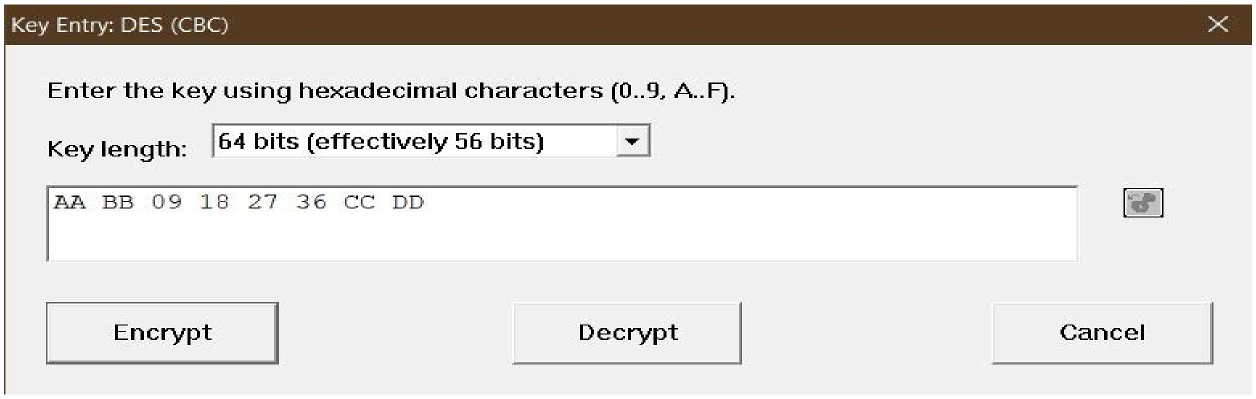
File Edit View Encrypt/Decrypt Digital Signatures/PKI Indiv. Procedures Analysis Options Window Help

Hello! My name is Prerak Mann.

DES

Simplified-DES The S-DES encryption algorithm takes an 8-bit block of plaintext (example: 10111101) and a 10-bit key as input and produces an 8-bit block of ciphertext as output. The S-DES decryption algorithm takes an 8-bit block of ciphertext and the same 10-bit key used to produce that ciphertext as input and produces the original 8-bit block of plaintext.

Key

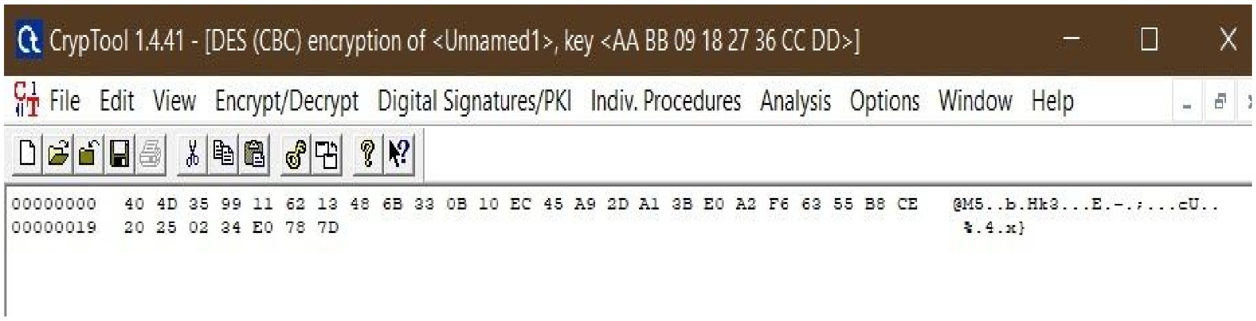


Enter the key using hexadecimal characters (0..9, A..F).

Key length: 64 bits (effectively 56 bits)

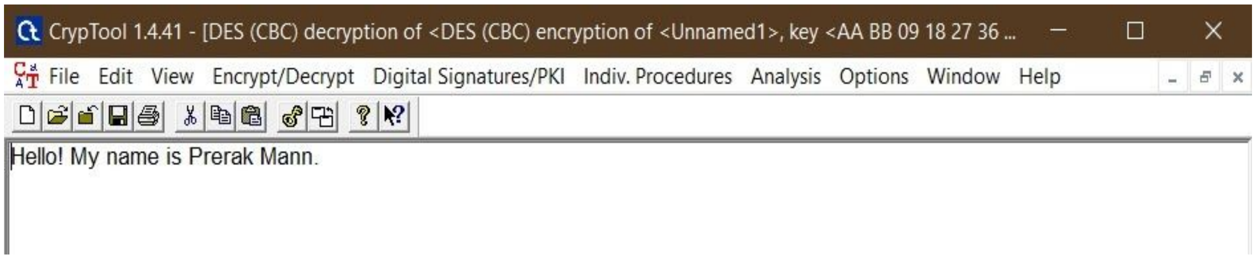
```
AA BB 09 18 27 36 CC DD
```

Encryption


CrypTool 1.4.41 - [DES (CBC) encryption of <Unnamed1>, key <AA BB 09 18 27 36 CC DD>]
File Edit View Encrypt/Decrypt Digital Signatures/PKI Indiv. Procedures Analysis Options Window Help

```
00000000   40 4D 35 99 11 62 13 48 6B 33 0B 10 EC 45 A9 2D A1 3B E0 A2 F6 63 55 B8 CE   @M5..b.Hk3...E.-.;...cU..
00000019   20 25 02 34 E0 78 7D                                                          %.4.x}
```

Decryption

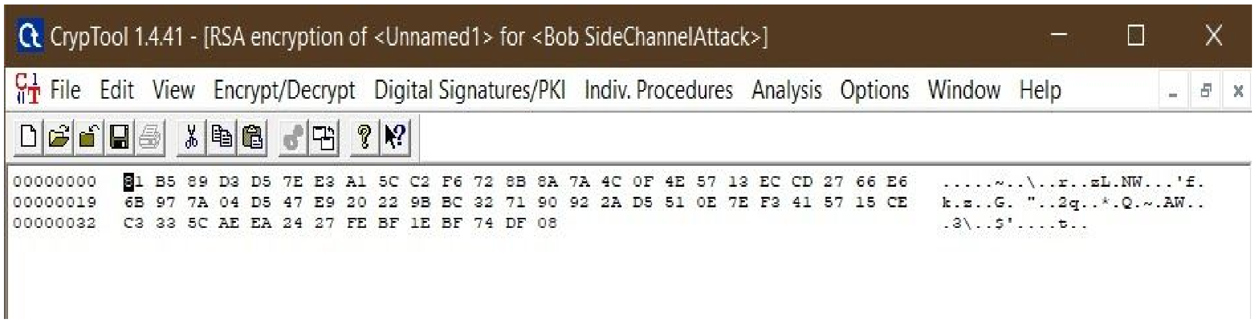

CrypTool 1.4.41 - [DES (CBC) decryption of <DES (CBC) encryption of <Unnamed1>, key <AA BB 09 18 27 36 ...
File Edit View Encrypt/Decrypt Digital Signatures/PKI Indiv. Procedures Analysis Options Window Help

Hello! My name is Prerak Mann.

RSA

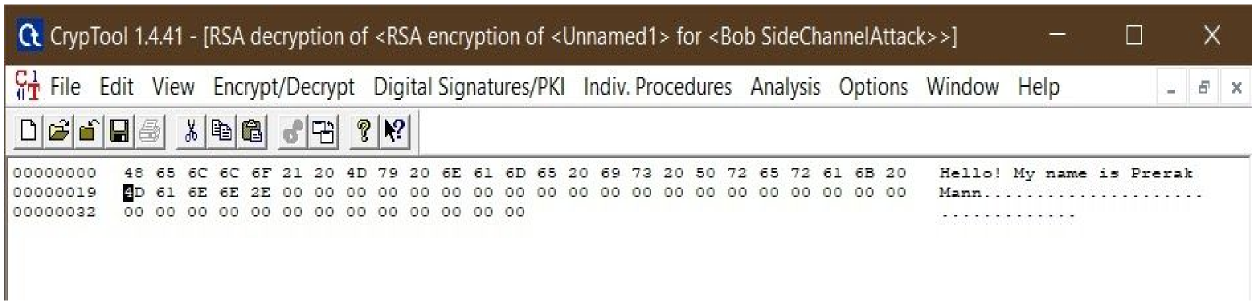
RSA is a cryptosystem for public-key encryption, and is widely used for securing sensitive data, particularly when being sent over an insecure network such as the Internet. Public-key cryptography, also known as asymmetric cryptography, uses two different but mathematically linked keys, one public and one private. The public key can be shared with everyone, whereas the private key must be kept secret.

Encryption


CrypTool 1.4.41 - [RSA encryption of <Unnamed1> for <Bob SideChannelAttack>]
File Edit View Encrypt/Decrypt Digital Signatures/PKI Indiv. Procedures Analysis Options Window Help

```
00000000   81 B5 89 D3 D5 7E E3 A1 5C C2 F6 72 8B 8A 7A 4C 0F 4E 57 13 EC CD 27 66 E6   .....~..\..r..zL.NW...'f.
00000019   6B 97 7A 04 D5 47 E9 20 22 9B BC 32 71 90 92 2A D5 51 0E 7E F3 41 57 15 CE   k.z..G. "..2q..*.Q.~.AW..
00000032   C3 33 5C AE EA 24 27 FE BF 1E BF 74 DF 08                                    .3\..$'....t..
```

Decryption


CrypTool 1.4.41 - [RSA decryption of <RSA encryption of <Unnamed1> for <Bob SideChannelAttack>>]
File Edit View Encrypt/Decrypt Digital Signatures/PKI Indiv. Procedures Analysis Options Window Help

```
00000000   48 65 6C 6C 6F 21 20 4D 79 20 6E 61 6D 65 20 69 73 20 50 72 65 72 61 6B 20   Hello! My name is Prerak
00000019   4D 61 6E 6E 2E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   Mann.....................
00000032   00 00 00 00 00 00 00 00 00 00 00 00 00                                       .............
```

Learning Outcomes:

Here we have used CrypTool Online to implement 5 different encryption/decryption algorithms - Hill Cipher, Caesar Cipher, AutoKey Cipher, Beaufort Cipher, Rotation Cipher.

Experiment 12

Aim: Study and use the Wireshark for the various network protocols

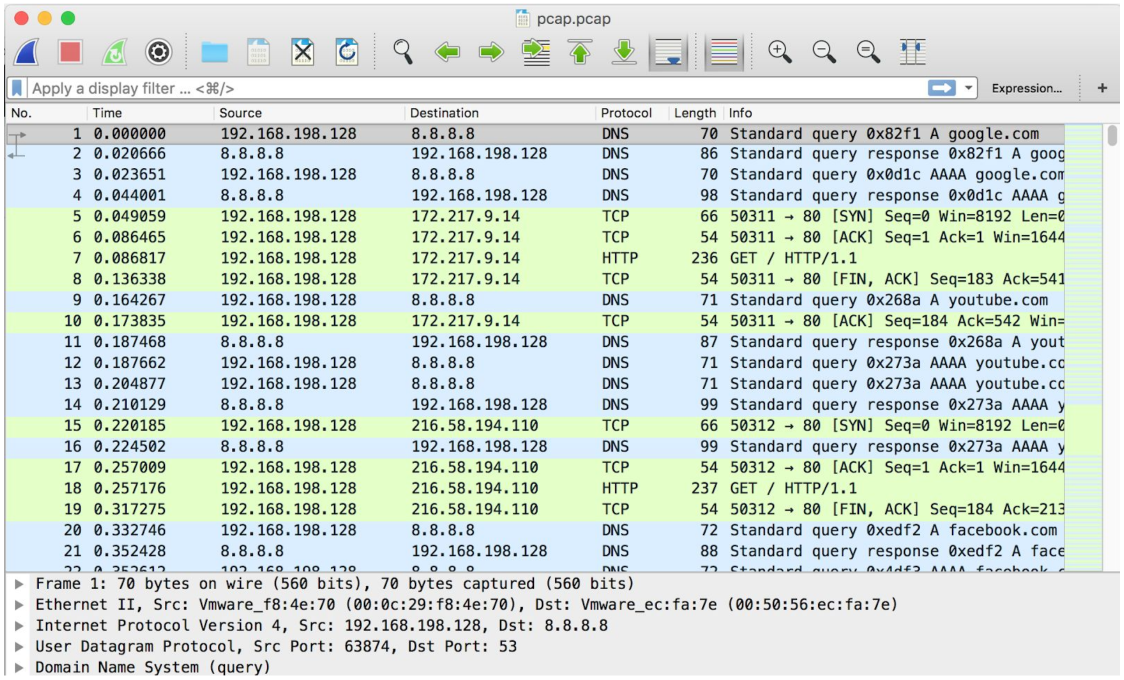
Theory: Wireshark is a packet sniffer and analysis tool. It captures network traffic on the local network and stores that data for offline analysis. It is used for network troubleshooting, analysis, software and communications protocol development, and education. Wireshark captures network traffic from Ethernet, Bluetooth, Wireless (IEEE.802.11), Token Ring, Frame Relay connections, and more.

- A “packet” is a single message from any network protocol (i.e., TCP, DNS, etc.)
- LAN traffic is in broadcast mode, meaning a single computer with Wireshark can see traffic between two other computers. If you want to see traffic to an external site, you need to capture the packets on the local computer.

Wireshark allows you to filter the log either before the capture starts or during analysis, so you can narrow down and zero into what you are looking for in the network trace. For example, you can set a filter to see TCP traffic between two IP addresses. You can set it only to show you the packets sent from one computer. The filters in Wireshark are one of the primary reasons it became the standard tool for packet analysis

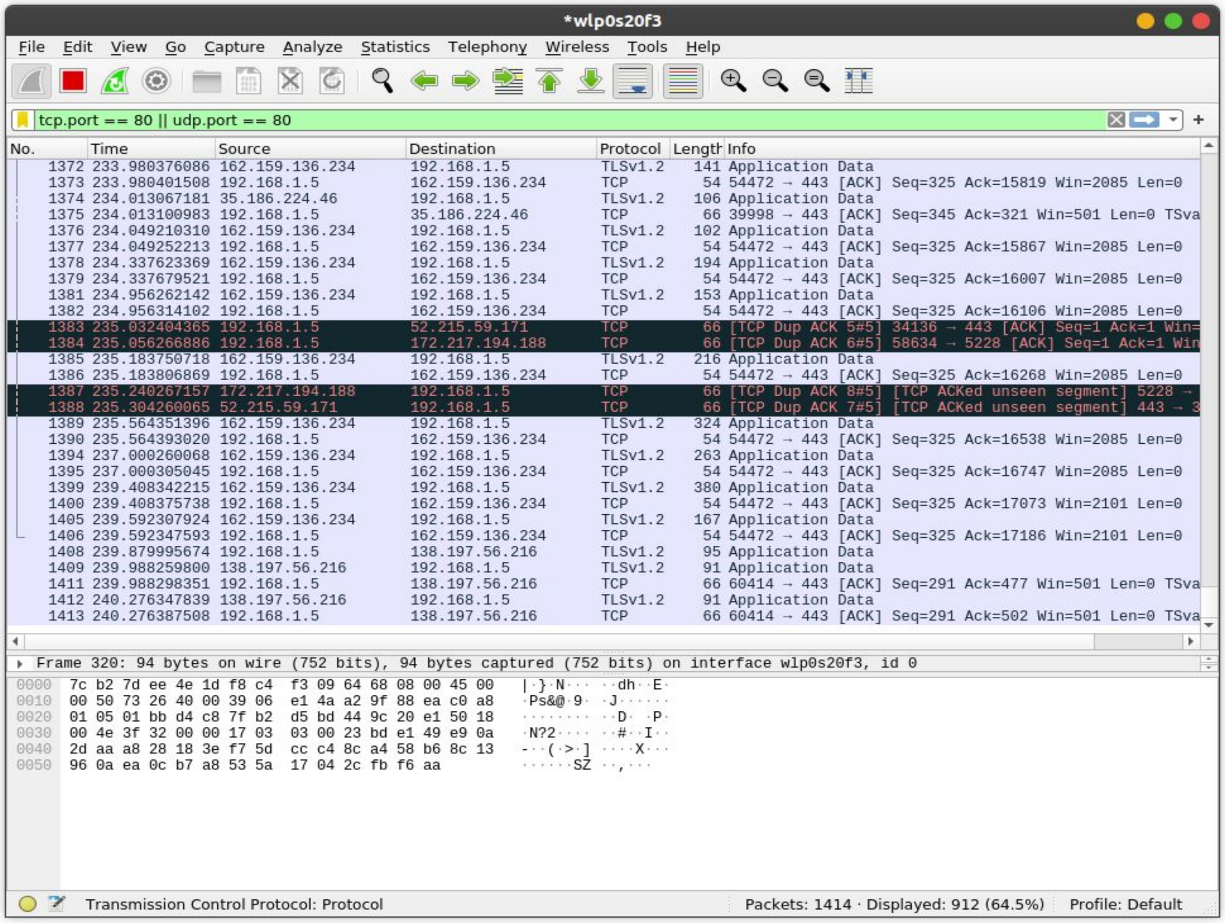
Output:

HTTP protocol



```
No.   Time       Source           Destination      Protocol  Length  Info
  1 0.000000   192.168.198.128  8.8.8.8          DNS       70  Standard query 0x82f1 A google.com
  2 0.020666   8.8.8.8          192.168.198.128  DNS       86  Standard query response 0x82f1 A goog
  3 0.023651   192.168.198.128  8.8.8.8          DNS       70  Standard query 0x0d1c AAAA google.com
  4 0.044001   8.8.8.8          192.168.198.128  DNS       98  Standard query response 0x0d1c AAAA g
  5 0.049059   192.168.198.128  172.217.9.14     TCP       66  50311 → 80 [SYN] Seq=0 Win=8192 Len=0
  6 0.086465   192.168.198.128  172.217.9.14     TCP       54  50311 → 80 [ACK] Seq=1 Ack=1 Win=1644
  7 0.086817   192.168.198.128  172.217.9.14     HTTP     236  GET / HTTP/1.1
  8 0.136338   192.168.198.128  172.217.9.14     TCP       54  50311 → 80 [FIN, ACK] Seq=183 Ack=541
  9 0.164267   192.168.198.128  8.8.8.8          DNS       71  Standard query 0x268a A youtube.com
 10 0.173835   192.168.198.128  172.217.9.14     TCP       54  50311 → 80 [ACK] Seq=184 Ack=542 Win=
 11 0.187468   8.8.8.8          192.168.198.128  DNS       87  Standard query response 0x268a A yout
 12 0.187662   192.168.198.128  8.8.8.8          DNS       71  Standard query 0x273a AAAA youtube.co
 13 0.204877   192.168.198.128  8.8.8.8          DNS       71  Standard query 0x273a AAAA youtube.co
 14 0.210129   8.8.8.8          192.168.198.128  DNS       99  Standard query response 0x273a AAAA y
 15 0.220185   192.168.198.128  216.58.194.110   TCP       66  50312 → 80 [SYN] Seq=0 Win=8192 Len=0
 16 0.224502   8.8.8.8          192.168.198.128  DNS       99  Standard query response 0x273a AAAA y
 17 0.257009   192.168.198.128  216.58.194.110   TCP       54  50312 → 80 [ACK] Seq=1 Ack=1 Win=1644
 18 0.257176   192.168.198.128  216.58.194.110   HTTP     237  GET / HTTP/1.1
 19 0.317275   192.168.198.128  216.58.194.110   TCP       54  50312 → 80 [FIN, ACK] Seq=184 Ack=213
 20 0.332746   192.168.198.128  8.8.8.8          DNS       72  Standard query 0xedf2 A facebook.com
 21 0.352428   8.8.8.8          192.168.198.128  DNS       88  Standard query response 0xedf2 A face
 22 0.352612   192.168.198.128  8.8.8.8          DNS       72  Standard query 0x4df3 AAAA facebook.c
▶ Frame 1: 70 bytes on wire (560 bits), 70 bytes captured (560 bits)
▶ Ethernet II, Src: Vmware_f8:4e:70 (00:0c:29:f8:4e:70), Dst: Vmware_ec:fa:7e (00:50:56:ec:fa:7e)
▶ Internet Protocol Version 4, Src: 192.168.198.128, Dst: 8.8.8.8
▶ User Datagram Protocol, Src Port: 63874, Dst Port: 53
▶ Domain Name System (query)
```

TCP protocol


*wlp0s20f3
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tcp.port == 80 || udp.port == 80
```

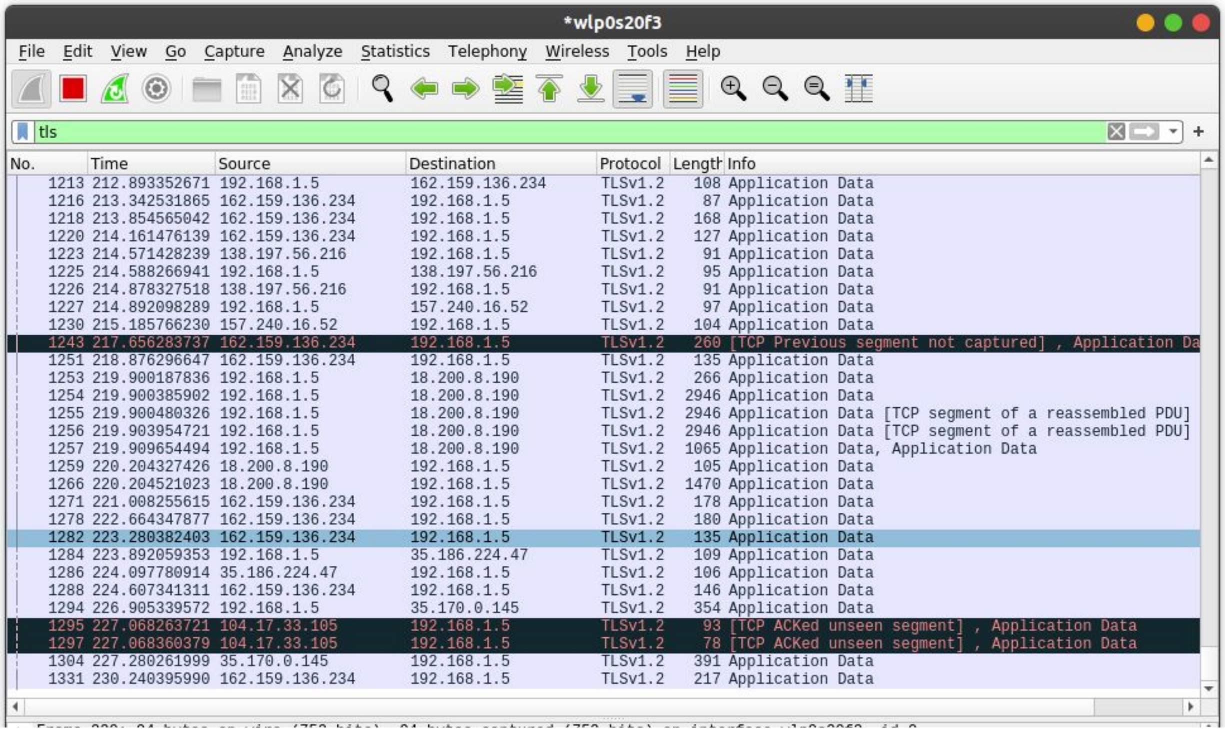
```
No.  Time         Source          Destination     Protocol Length Info
1372 233.980376086 162.159.136.234 192.168.1.5     TLSv1.2  141 Application Data
1373 233.980401508 192.168.1.5     162.159.136.234 TCP       54 54472 → 443 [ACK] Seq=325 Ack=15819 Win=2085 Len=0
1374 234.013067181 35.186.224.46   192.168.1.5     TLSv1.2  106 Application Data
1375 234.013100983 192.168.1.5     35.186.224.46   TCP       66 39998 → 443 [ACK] Seq=345 Ack=321 Win=501 Len=0 TSva
1376 234.049210310 162.159.136.234 192.168.1.5     TLSv1.2  102 Application Data
1377 234.049252213 192.168.1.5     162.159.136.234 TCP       54 54472 → 443 [ACK] Seq=325 Ack=15867 Win=2085 Len=0
1378 234.337623369 162.159.136.234 192.168.1.5     TLSv1.2  194 Application Data
1379 234.337679521 192.168.1.5     162.159.136.234 TCP       54 54472 → 443 [ACK] Seq=325 Ack=16007 Win=2085 Len=0
1381 234.956262142 162.159.136.234 192.168.1.5     TLSv1.2  153 Application Data
1382 234.956314102 192.168.1.5     162.159.136.234 TCP       54 54472 → 443 [ACK] Seq=325 Ack=16106 Win=2085 Len=0
1383 235.032404365 192.168.1.5     52.215.59.171   TCP       66 [TCP Dup ACK 5#5] 34136 → 443 [ACK] Seq=1 Ack=1 Win=
1384 235.056266886 192.168.1.5     172.217.194.188 TCP       66 [TCP Dup ACK 6#5] 58634 → 5228 [ACK] Seq=1 Ack=1 Win
1385 235.183750718 162.159.136.234 192.168.1.5     TLSv1.2  216 Application Data
1386 235.183806869 192.168.1.5     162.159.136.234 TCP       54 54472 → 443 [ACK] Seq=325 Ack=16268 Win=2085 Len=0
1387 235.240267157 172.217.194.188 192.168.1.5     TCP       66 [TCP Dup ACK 8#5] [TCP ACKed unseen segment] 5228 →
1388 235.304260065 52.215.59.171   192.168.1.5     TCP       66 [TCP Dup ACK 7#5] [TCP ACKed unseen segment] 443 → 3
1389 235.564351396 162.159.136.234 192.168.1.5     TLSv1.2  324 Application Data
1390 235.564393020 192.168.1.5     162.159.136.234 TCP       54 54472 → 443 [ACK] Seq=325 Ack=16538 Win=2085 Len=0
1394 237.000260068 162.159.136.234 192.168.1.5     TLSv1.2  263 Application Data
1395 237.000305045 192.168.1.5     162.159.136.234 TCP       54 54472 → 443 [ACK] Seq=325 Ack=16747 Win=2085 Len=0
1399 239.408342215 162.159.136.234 192.168.1.5     TLSv1.2  380 Application Data
1400 239.408375738 192.168.1.5     162.159.136.234 TCP       54 54472 → 443 [ACK] Seq=325 Ack=17073 Win=2101 Len=0
1405 239.592307924 162.159.136.234 192.168.1.5     TLSv1.2  167 Application Data
1406 239.592347593 192.168.1.5     162.159.136.234 TCP       54 54472 → 443 [ACK] Seq=325 Ack=17186 Win=2101 Len=0
1408 239.879995674 192.168.1.5     138.197.56.216  TLSv1.2   95 Application Data
1409 239.988259800 138.197.56.216  192.168.1.5     TLSv1.2   91 Application Data
1411 239.988298351 192.168.1.5     138.197.56.216  TCP       66 60414 → 443 [ACK] Seq=291 Ack=477 Win=501 Len=0 TSva
1412 240.276347839 138.197.56.216  192.168.1.5     TLSv1.2   91 Application Data
1413 240.276387508 192.168.1.5     138.197.56.216  TCP       66 60414 → 443 [ACK] Seq=291 Ack=502 Win=501 Len=0 TSva
```

```
▸ Frame 320: 94 bytes on wire (752 bits), 94 bytes captured (752 bits) on interface wlp0s20f3, id 0
```

```
0000  7c b2 7d ee 4e 1d f8 c4  f3 09 64 68 08 00 45 00   |·}·N···  ··dh··E·
0010  00 50 73 26 40 00 39 06  e1 4a a2 9f 88 ea c0 a8   ·Ps&@·9·  ·J······
0020  01 05 01 bb d4 c8 7f b2  d5 bd 44 9c 20 e1 50 18   ········  ··D· ·P·
0030  00 4e 3f 32 00 00 17 03  03 00 23 bd e1 49 e9 0a   ·N?2····  ··#··I··
0040  2d aa a8 28 18 3e f7 5d  cc c4 8c a4 58 b6 8c 13   -··(·>·]  ····X···
0050  96 0a ea 0c b7 a8 53 5a  17 04 2c fb f6 aa         ······SZ  ··,···
```

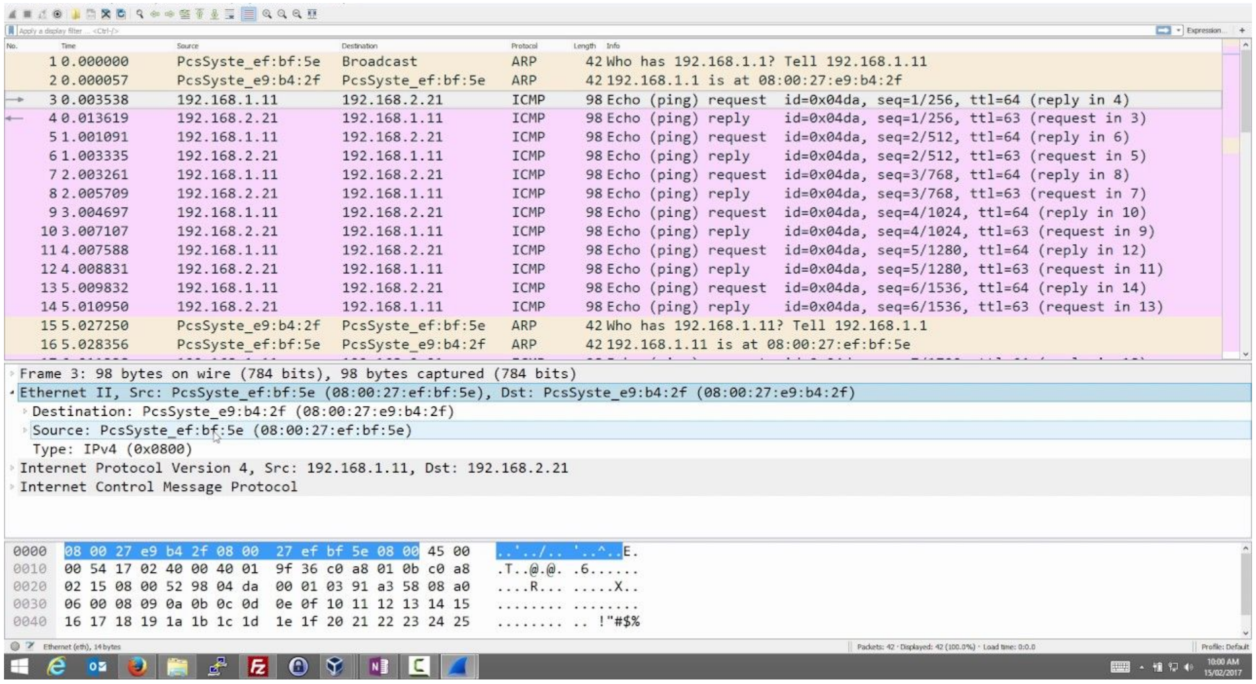
```
Transmission Control Protocol: Protocol          Packets: 1414 · Displayed: 912 (64.5%)    Profile: Default
```

TLS Protocol


*wlp0s20f3
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tls

```
No.   Time          Source           Destination     Protocol Length Info
 1213 212.893352671 192.168.1.5      162.159.136.234 TLSv1.2   108 Application Data
 1216 213.342531865 162.159.136.234  192.168.1.5     TLSv1.2    87 Application Data
 1218 213.854565042 162.159.136.234  192.168.1.5     TLSv1.2   168 Application Data
 1220 214.161476139 162.159.136.234  192.168.1.5     TLSv1.2   127 Application Data
 1223 214.571428239 138.197.56.216   192.168.1.5     TLSv1.2    91 Application Data
 1225 214.588266941 192.168.1.5      138.197.56.216  TLSv1.2    95 Application Data
 1226 214.878327518 138.197.56.216   192.168.1.5     TLSv1.2    91 Application Data
 1227 214.892098289 192.168.1.5      157.240.16.52   TLSv1.2    97 Application Data
 1230 215.185766230 157.240.16.52    192.168.1.5     TLSv1.2   104 Application Data
 1243 217.656283737 162.159.136.234  192.168.1.5     TLSv1.2   260 [TCP Previous segment not captured] , Application Da
 1251 218.876296647 162.159.136.234  192.168.1.5     TLSv1.2   135 Application Data
 1253 219.900187836 192.168.1.5      18.200.8.190    TLSv1.2   266 Application Data
 1254 219.900385902 192.168.1.5      18.200.8.190    TLSv1.2  2946 Application Data
 1255 219.900480326 192.168.1.5      18.200.8.190    TLSv1.2  2946 Application Data [TCP segment of a reassembled PDU]
 1256 219.903954721 192.168.1.5      18.200.8.190    TLSv1.2  2946 Application Data [TCP segment of a reassembled PDU]
 1257 219.909654494 192.168.1.5      18.200.8.190    TLSv1.2  1065 Application Data, Application Data
 1259 220.204327426 18.200.8.190     192.168.1.5     TLSv1.2   105 Application Data
 1266 220.204521023 18.200.8.190     192.168.1.5     TLSv1.2  1470 Application Data
 1271 221.008255615 162.159.136.234  192.168.1.5     TLSv1.2   178 Application Data
 1278 222.664347877 162.159.136.234  192.168.1.5     TLSv1.2   180 Application Data
 1282 223.280382403 162.159.136.234  192.168.1.5     TLSv1.2   135 Application Data
 1284 223.892059353 192.168.1.5      35.186.224.47   TLSv1.2   109 Application Data
 1286 224.097780914 35.186.224.47    192.168.1.5     TLSv1.2   106 Application Data
 1288 224.607341311 162.159.136.234  192.168.1.5     TLSv1.2   146 Application Data
 1294 226.905339572 192.168.1.5      35.170.0.145    TLSv1.2   354 Application Data
 1295 227.068263721 104.17.33.105    192.168.1.5     TLSv1.2    93 [TCP ACKed unseen segment] , Application Data
 1297 227.068360379 104.17.33.105    192.168.1.5     TLSv1.2    78 [TCP ACKed unseen segment] , Application Data
 1304 227.280261999 35.170.0.145     192.168.1.5     TLSv1.2   391 Application Data
 1331 230.240395990 162.159.136.234  192.168.1.5     TLSv1.2   217 Application Data
```

ICMP protocol

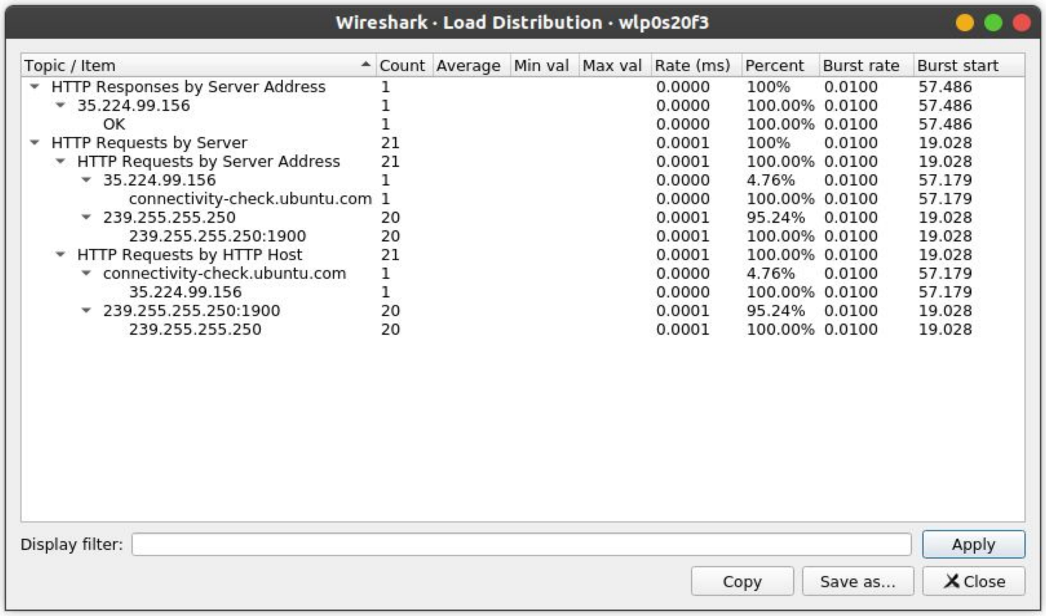


```
No. Time      Source             Destination        Protocol Length Info
  1 0.000000  PcsSyste_ef:bf:5e  Broadcast          ARP      42 Who has 192.168.1.1? Tell 192.168.1.11
  2 0.000057  PcsSyste_e9:b4:2f  PcsSyste_ef:bf:5e  ARP      42 192.168.1.1 is at 08:00:27:e9:b4:2f
  3 0.003538  192.168.1.11       192.168.2.21       ICMP     98 Echo (ping) request  id=0x04da, seq=1/256, ttl=64 (reply in 4)
  4 0.013619  192.168.2.21       192.168.1.11       ICMP     98 Echo (ping) reply    id=0x04da, seq=1/256, ttl=63 (request in 3)
  5 1.001091  192.168.1.11       192.168.2.21       ICMP     98 Echo (ping) request  id=0x04da, seq=2/512, ttl=64 (reply in 6)
  6 1.003335  192.168.2.21       192.168.1.11       ICMP     98 Echo (ping) reply    id=0x04da, seq=2/512, ttl=63 (request in 5)
  7 2.003261  192.168.1.11       192.168.2.21       ICMP     98 Echo (ping) request  id=0x04da, seq=3/768, ttl=64 (reply in 8)
  8 2.005709  192.168.2.21       192.168.1.11       ICMP     98 Echo (ping) reply    id=0x04da, seq=3/768, ttl=63 (request in 7)
  9 3.004697  192.168.1.11       192.168.2.21       ICMP     98 Echo (ping) request  id=0x04da, seq=4/1024, ttl=64 (reply in 10)
 10 3.007107  192.168.2.21       192.168.1.11       ICMP     98 Echo (ping) reply    id=0x04da, seq=4/1024, ttl=63 (request in 9)
 11 4.007588  192.168.1.11       192.168.2.21       ICMP     98 Echo (ping) request  id=0x04da, seq=5/1280, ttl=64 (reply in 12)
 12 4.008831  192.168.2.21       192.168.1.11       ICMP     98 Echo (ping) reply    id=0x04da, seq=5/1280, ttl=63 (request in 11)
 13 5.009832  192.168.1.11       192.168.2.21       ICMP     98 Echo (ping) request  id=0x04da, seq=6/1536, ttl=64 (reply in 14)
 14 5.010950  192.168.2.21       192.168.1.11       ICMP     98 Echo (ping) reply    id=0x04da, seq=6/1536, ttl=63 (request in 13)
 15 5.027250  PcsSyste_e9:b4:2f  PcsSyste_ef:bf:5e  ARP      42 Who has 192.168.1.11? Tell 192.168.1.1
 16 5.028356  PcsSyste_ef:bf:5e  PcsSyste_e9:b4:2f  ARP      42 192.168.1.11 is at 08:00:27:ef:bf:5e
```

```
Frame 3: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
Ethernet II, Src: PcsSyste_ef:bf:5e (08:00:27:ef:bf:5e), Dst: PcsSyste_e9:b4:2f (08:00:27:e9:b4:2f)
  Destination: PcsSyste_e9:b4:2f (08:00:27:e9:b4:2f)
  Source: PcsSyste_ef:bf:5e (08:00:27:ef:bf:5e)
  Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 192.168.1.11, Dst: 192.168.2.21
Internet Control Message Protocol
```

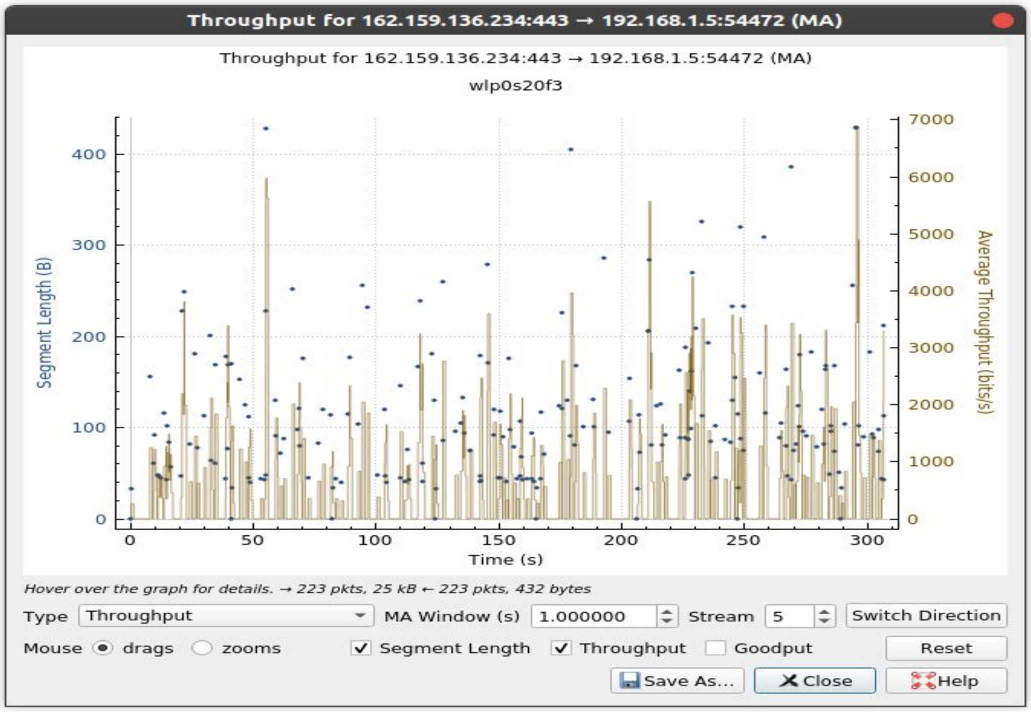
```
0000  08 00 27 e9 b4 2f 08 00  27 ef bf 5e 08 00 45 00   ..'../.. '..^..E.
0010  00 54 17 02 40 00 40 01  9f 36 c0 a8 01 0b c0 a8   .T..@.@. .6......
0020  02 15 08 00 52 98 04 da  00 01 03 91 a3 58 08 a0   ....R... .....X..
0030  06 00 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ........ ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
```

Analysing HTTP packets



Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
HTTP Responses by Server Address	1				0.0000	100%	0.0100	57.486
35.224.99.156	1				0.0000	100.00%	0.0100	57.486
OK	1				0.0000	100.00%	0.0100	57.486
HTTP Requests by Server	21				0.0001	100%	0.0100	19.028
HTTP Requests by Server Address	21				0.0001	100.00%	0.0100	19.028
35.224.99.156	1				0.0000	4.76%	0.0100	57.179
connectivity-check.ubuntu.com	1				0.0000	100.00%	0.0100	57.179
239.255.255.250	20				0.0001	95.24%	0.0100	19.028
239.255.255.250:1900	20				0.0001	100.00%	0.0100	19.028
HTTP Requests by HTTP Host	21				0.0001	100.00%	0.0100	19.028
connectivity-check.ubuntu.com	1				0.0000	4.76%	0.0100	57.179
35.224.99.156	1				0.0000	100.00%	0.0100	57.179
239.255.255.250:1900	20				0.0001	95.24%	0.0100	19.028
239.255.255.250	20				0.0001	100.00%	0.0100	19.028

Analysing TCP packets



Hover over the graph for details. → 223 pkts, 25 kB ← 223 pkts, 432 bytes

Type: Throughput — MA Window (s): 1.000000 — Stream: 5

Mouse: (●) drags ( ) zooms — [✓] Segment Length — [✓] Throughput — [ ] Goodput

Learning Outcomes:

Here we learned how to use wireshark for packet capturing using different protocol filters and how to analyze these packets.