EXPERIMENT – 1(a): CAESAR CIPHER

AIM: To implement a program for encrypting a plain text and decrypting a cipher text using Caesar Cipher (shift cipher) substitution technique.

Requirements: Java Runtime Environment and Java Development Kit.

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 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 positions 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:

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

(Encryption Phase with shift n)

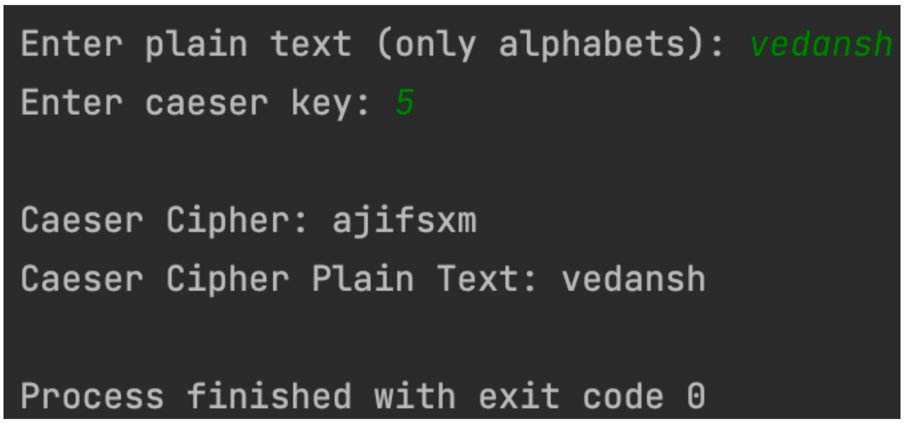
$$D_n(x) = (x - n) mod\ 26$$

(Decryption Phase with shift n)

Program:

```
import java.util.Scanner;
public class INS_Lab01_A {
    private static final String alphabet = "abcdefghijklmnopqrstuvwxyz";
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter plain text (only alphabets): ");
        String plainText = sc.next().toLowerCase();
        System.out.print("Enter caeser key: ");
        int key = sc.nextInt();
        String caeserEncrypted = caeserCipherEncrypt(plainText,key);
        String caeserDecrypted = caeserCipherDecrypt(caeserEncrypted, key);
        System.out.println("\nCaeser Cipher: "+caeserEncrypted);
        System.out.println("Caeser Cipher Plain Text: "+caeserDecrypted);
    }
    public static String caeserCipherEncrypt(String plainText, int key) {
        String encrypted = "";
        for(char ch: plainText.toCharArray())
            encrypted += alphabet.charAt((alphabet.indexOf(ch) + key)%26);
        return encrypted;
    }
    public static String caeserCipherDecrypt(String encrypted, int key) {
        String decrypted = "";
        for(char ch: encrypted.toCharArray())
            decrypted += alphabet.charAt((alphabet.indexOf(ch) - key + 26)%26);
        return decrypted;
    }
}
```

Output :



```
Enter plain text (only alphabets): vedansh
Enter caeser key: 5

Caeser Cipher: ajifsxm
Caeser Cipher Plain Text: vedansh

Process finished with exit code 0
```

Result: Thus the program to implement caesar cipher encryption technique was developed and executed.

EXPERIMENT – 1(b): PLAYFAIR CIPHER

AIM: To implement a program for encrypting a plain text and decrypting a cipher text using Playfair Cipher substitution technique.

Requirements: Java Runtime Environment and Java Development Kit.

Theory:

The Playfair Cipher Encryption 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.

   Rules:

   - Pair cannot be made with same letter. Break the letter in single and add a bogus letter to the previous letter.

     Plain Text: “helloe”
     After Split: ‘he’ ‘lx’ ‘lo’ ‘ez’

   - If the letter is standing alone in the process of pairing, then add an extra bogus letter with the alone letter.

     Plain Text: “hello”
     After Split: ‘he’ ‘lx’ ‘lo’

RULES FOR ENCRYPTION:

- If both the letters are in the same column: Take the letter below each one (going back to the top if at the bottom).
- If both the letters are in the same row: Take the letter to the right of each one (going back to the leftmost if at the rightmost position).
- If neither of the above rules is true: Form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

RULES FOR DECRYPTION :

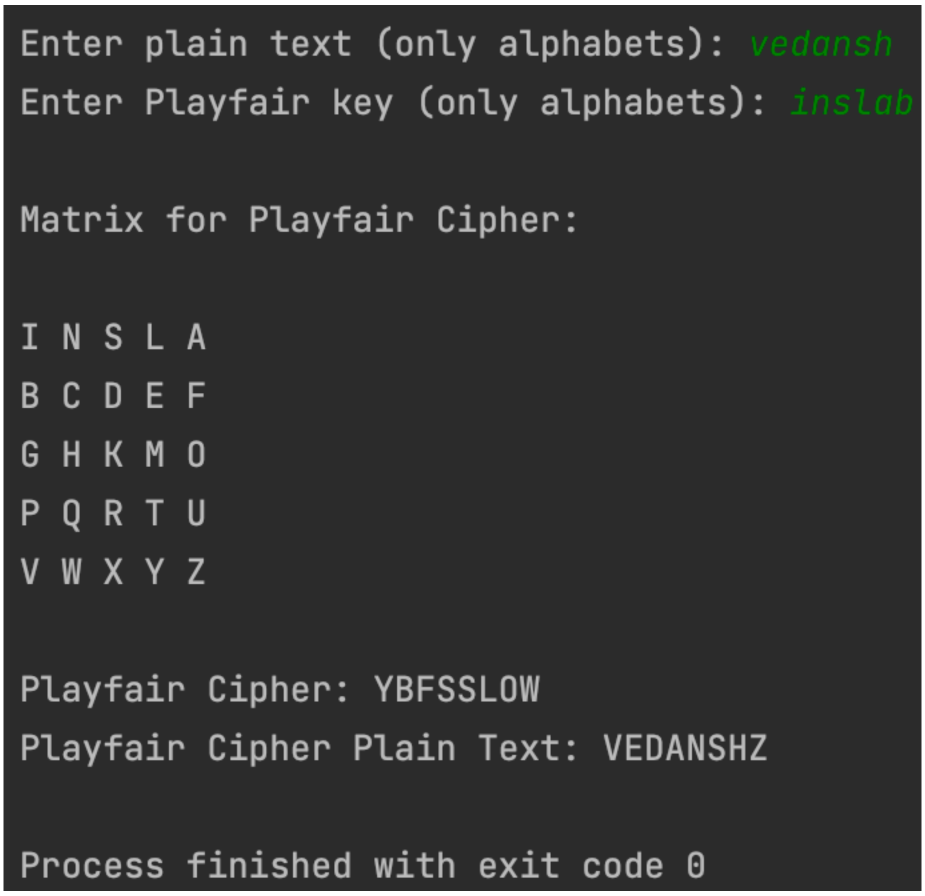
- Decrypting the Playfair cipher is as simple as doing the same process in reverse. The receiver has the same key and can create the same key table, and then decrypt any messages made using that key.
- If both the letters are in the same column: Take the letter above each one (going back to the bottom if at the top).
- If both the letters are in the same row: Take the letter to the left of each one (going back to the rightmost if at the leftmost position).
- If neither of the above rules is true: Form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

Program:

```
import java.util.Scanner;
public class INS_Lab01_B {
  private static String matrix;
  private static String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter plain text (only alphabets): ");
    String plainText = sc.next().toUpperCase();
    System.out.print("Enter Playfair key (only alphabets): ");
    String key = sc.next().toUpperCase();
    String playfairEncrypted = playfairCipherEncrypt(plainText,key);
    String playfairDecrypted = playfairCipherDecrypt(playfairEncrypted);
    System.out.println("\n\nPlayfair Cipher: "+playfairEncrypted);
    System.out.println("Playfair Cipher Plain Text: "+playfairDecrypted);
  }
  public static String playfairCipherEncrypt(String plainText, String key) {
    String encrypted = "";
    matrix = "";
    for(int i = 0; i < key.length(); i++)
      if(matrix.indexOf(key.charAt(i)) == -1)
        matrix += key.charAt(i);
    for(int i = 0; i < alphabet.length(); i++)
      if(matrix.indexOf(alphabet.charAt(i)) == -1)
        matrix += alphabet.charAt(i);
    plainText = plainText.replace("J", "I");
    matrix = matrix.replace("J", "I");
    matrix = matrix.substring(0, matrix.lastIndexOf("I")) +
matrix.substring(matrix.lastIndexOf("I")+1);
    System.out.println("\nMatrix for Playfair Cipher: ");
    for(int i = 0; i < 25; i++) {
      if(i%5 == 0)
        System.out.println();
      System.out.print(matrix.charAt(i)+ " ");
    }
    char a,b;
    while(plainText.length() > 0) {
      if(plainText.length() == 1) {
        if(plainText.charAt(0) == 'Z') {
            a = 'Z';
            b = 'X';
        }
        else {
            a = plainText.charAt(0);
            b = 'Z';
        }
        plainText = "";
      }
      else {
        if(plainText.charAt(0) == plainText.charAt(1)) {
            if(plainText.charAt(0) == 'X') {
                a = 'X';
                b = 'Z';
            }
            else {
                a = plainText.charAt(0);
                b = 'X';
            }
            plainText = plainText.substring(1);
        }
        else {
            a = plainText.charAt(0);
            b = plainText.charAt(1);
            plainText = plainText.substring(2);
        }
      }
      int a_row = matrix.indexOf(a)/5;
      int a_col = matrix.indexOf(a)%5;
      int b_row = matrix.indexOf(b)/5;
      int b_col = matrix.indexOf(b)%5;
      if(a_row == b_row)
        encrypted += Character.toString(matrix.charAt(a_row*5+(a_col+1)%5))
+ Character.toString(matrix.charAt(b_row*5+(b_col+1)%5));
      else if(a_col == b_col)
        encrypted +=
Character.toString(matrix.charAt((matrix.indexOf(a)+5)%25)) +
Character.toString(matrix.charAt((matrix.indexOf(b)+5)%25));
      else
```

```
            encrypted += Character.toString(matrix.charAt(a_row*5+b_col)) +
Character.toString(matrix.charAt(b_row*5+a_col));
        }
        return encrypted;
    }
    public static String playfairCipherDecrypt(String encrypted) {
        String decrypted = "";
        while(encrypted.length() > 0) {
            char a = encrypted.charAt(0);
            char b = encrypted.charAt(1);
            encrypted = encrypted.substring(2);
            int a_row = matrix.indexOf(a)/5;
            int a_col = matrix.indexOf(a)%5;
            int b_row = matrix.indexOf(b)/5;
            int b_col = matrix.indexOf(b)%5;
            if(a_row == b_row)
                decrypted += Character.toString(matrix.charAt(a_row*5+(4+a_col)%5))
+ Character.toString(matrix.charAt(b_row*5+(4+b_col)%5));
            else if(a_col == b_col)
                decrypted +=
Character.toString(matrix.charAt(((4+a_row)%5)*5+a_col)) +
Character.toString(matrix.charAt(((4+b_row)%5)*5+b_col));
            else
                decrypted += Character.toString(matrix.charAt(a_row*5+b_col)) +
Character.toString(matrix.charAt(b_row*5+a_col));
        }
        return decrypted;
    }
}
```

Output :



```
Enter plain text (only alphabets): vedansh
Enter Playfair key (only alphabets): inslab

Matrix for Playfair Cipher:

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

Playfair Cipher: YBFSSLOW
Playfair Cipher Plain Text: VEDANSHZ

Process finished with exit code 0
```

Result: Thus the program to implement playfair cipher encryption technique was developed and executed.

EXPERIMENT – 2(a): HILL CIPHER

AIM: To implement a program for encrypting a plain text and decrypting a cipher text using Hill Cipher (shift cipher) substitution technique.

Requirements: C++ or any other language.

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:

We have to encrypt the message 'ACT' (n=3).The key is 'GYBNQKURP' which can be written as the n x n matrix:

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix}$$

The message 'ACT' is written as vector:

$$\begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix}$$

The enciphered vector is given as:

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

which corresponds to ciphertext of ‘POH’

Decryption:

To decrypt the message, we turn the ciphertext back into a vector, then simply multiply by the inverse matrix of the key matrix (IFKVIVVMI in letters).The inverse of the matrix used in the previous example is:

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix}^{-1} \equiv \begin{bmatrix} 8 & 5 & 10 \\ 21 & 8 & 21 \\ 21 & 12 & 8 \end{bmatrix} \pmod{26}$$

For the previous Ciphertext ‘POH’:

$$\begin{bmatrix} 8 & 5 & 10 \\ 21 & 8 & 21 \\ 21 & 12 & 8 \end{bmatrix} \begin{bmatrix} 15 \\ 14 \\ 7 \end{bmatrix} \equiv \begin{bmatrix} 260 \\ 574 \\ 539 \end{bmatrix} \equiv \begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix} \pmod{26}$$

which gives us back ‘ACT’.

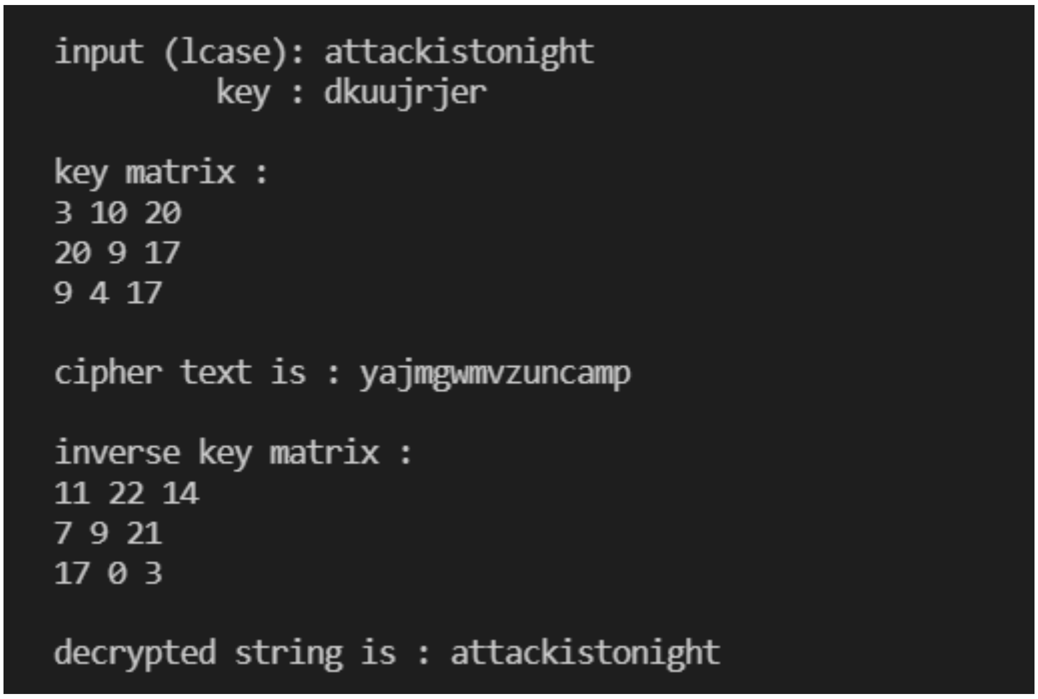
Program:

```
#include <iostream>
#include <math.h>
#include <vector>
#include <string>
using namespace std;
void printmat(int mat[3][3], int row, int col) {
   cout << endl;
   for (int i = 0; i < row; i++) {
      for (int j = 0; j < col; j++) {
         cout << " " << mat[i][j];
      }
      cout << endl;
   }
}
void printmat(int mat[3], int n) {
   cout << endl;
   for (int i = 0; i < n; i++) {
      cout << mat[i] << endl;
   }
}
int determinant(vector < vector < int >> mat, vector < int > col, int current_n, int
start_i, int start_j) {
   if (current_n == 1)
      return mat[start_i][start_j];
   int total = 0;
   for (int i = 0; i < col.size(); i++) {
      vector < int > subcol(col);
      subcol.erase(subcol.begin() + i);
      total += pow(-1, i) * mat[start_i][col[i]] * determinant(mat, subcol,
current_n - 1, start_i + 1, subcol[0]);
   }
   return total;
}
void inverse(int mat[3][3], int inmat[3][3], int n) {
   vector < int > temp;
   for (int i = 0; i < n; i++)
      temp.push_back(i);
   vector < vector < int >> adjoint_mat(n, vector < int > (n, 0));
```

```
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      adjoint_mat[i][j] = mat[i][j];
  int deter = determinant(adjoint_mat, temp, n, 0, 0);
  deter = (deter % 26);
  deter = (26 + deter) % 26;
  if (deter % 2 == 0 || deter == 13) {
    cout << "\n this key cant be used ,\n because it has no multiplicative inverse
";
    exit(0);
  } else {
    for (int i = 1; i < 26; i++) {
      if ((deter * i) % 26 == 1) {
        deter = i;
        break;
      }
    }
  }
  temp.pop_back();
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      vector < vector < int >> adjoint_copy(adjoint_mat);
      adjoint_copy.erase(adjoint_copy.begin() + i);
      for (int k = 0; k < adjoint_copy.size(); k++)
        adjoint_copy[k].erase(adjoint_copy[k].begin() + j);
      inmat[i][j] = determinant(adjoint_copy, temp, n - 1, 0, 0);
      inmat[i][j] *= pow(-1, i + j);
    }
  }
  for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
      int temp = inmat[i][j];
      inmat[i][j] = inmat[j][i];
      inmat[j][i] = temp;
    }
  }
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      inmat[i][j] = (inmat[i][j] * deter) % 26;
```

```
            inmat[i][j] = (26 + inmat[i][j]) % 26;
        }
    }
}
string getstring(int mat[3][3], int mat2[3], int resultant[3], int n) {
    string res = "";
    for (int i = 0; i < n; i++) {
        resultant[i] = 0;
        for (int j = 0; j < n; j++)
            resultant[i] += (mat[i][j] * mat2[j]);
        resultant[i] %= 26;
        res += (char) resultant[i] + 97;
    }
    return res;
}
int main() {
    system("cls");
    int matrix[3][3] = {{3, 4, 2},{2, 1, -2},{0, 1, 1}};
    int inversemat[3][3];
    int input_mat[3];
    int size = 2;
    string input = "batman";
    string key = "kdbt";
    string cipher = "", decipher = "";
    int decipher_mat[3], cipher_mat[3];
    cout << "\n input (lcase): " << input;
    cout << "\n          key : " << key;
    cout << "\n         size : " << size;
    int count = 0;
    for (int i = 0; i < size; i++)
        for (int j = 0; j < size; j++)
            matrix[i][j] = key[count++] - 97;
    cout << "\n\n key matrix :";
    printmat(matrix, size, size);
    inverse(matrix, inversemat, size);
    for (int a = 0; a < input.size() / size; a++) {
        for (int i = 0; i < size; i++)
            input_mat[i] = input[(size * a) + i] - 97;
        cipher += getstring(matrix, input_mat, cipher_mat, size);
    }
    cout << " \n cipher text is : " << cipher;
    cout << "\n\n inverse key matrix :";
    printmat(inversemat, size, size);
    for (int a = 0; a < cipher.size() / size; a++) {
        for (int i = 0; i < size; i++)
            cipher_mat[i] = cipher[(size * a) + i] - 97;
        decipher += getstring(inversemat, cipher_mat, decipher_mat, size);
    }
    cout << "\n decrypted string is : " << decipher << "\n\n";
    return 0;
}
```

Output :



```
input (lcase): attackistonight
         key : dkuujrjer

key matrix :
3 10 20
20 9 17
9 4 17

cipher text is : yajmgwmvzuncamp

inverse key matrix :
11 22 14
7 9 21
17 0 3

decrypted string is : attackistonight
```

Result: Thus the program to implement hill cipher encryption technique was developed and executed.

EXPERIMENT – 2(b): VIGENERE CIPHER

AIM: To implement a program for encrypting a plain text and decrypting a cipher text using Vigenere Cipher substitution technique.

Requirements: Java Runtime Environment and Java Development Kit.

Theory: Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of 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 Vigenere square or Vigenere 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.
- At different points in the encryption process, the cipher uses a different alphabet from one of the rows.
- The alphabet used at each point depends on a repeating keyword.

Example:

Input: Plaintext : GEEKSFORGEEKS

Keyword : AYUSH

Output: Ciphertext : GCYCZFMLYLEIM

For generating key, the given keyword is repeated in a circular manner until it matches the length of the plain text.

The keyword "AYUSH" generates the key "AYUSHAYUSHAYU"

The plain text is then encrypted using the process explained below.

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 Vigenere 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.

The plaintext(P) and key(K) are added modulo 26.

$E_i = (P_i + K_i) \bmod 26$

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.

A more easy implementation could be to visualize Vigenère algebraically by converting [A-Z] into numbers [0–25].

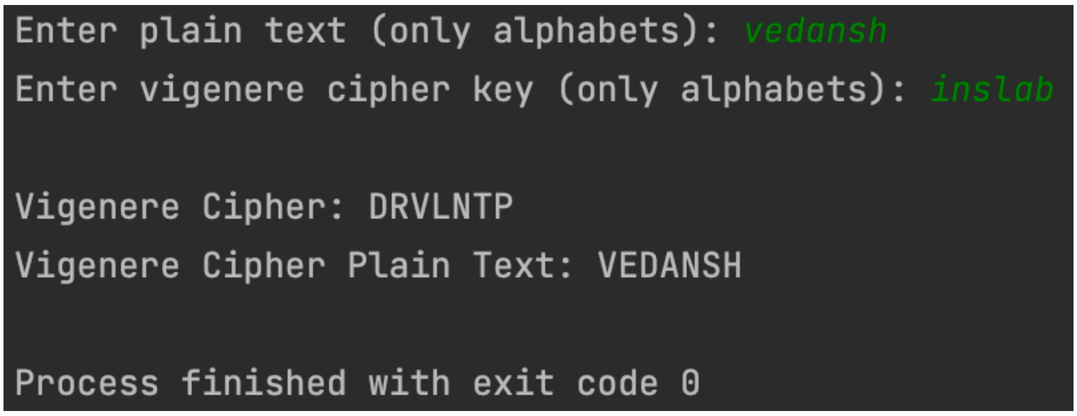
$D_i = (E_i - K_i + 26) \bmod 26$

Program:

```
import java.util.Scanner;
public class INS_Lab02_B {
   private static final String alphabets = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter plain text (only alphabets): ");
      String plainText = sc.nextLine().toUpperCase();
      System.out.print("Enter vigenere cipher key (only alphabets): ");
      String key = sc.nextLine().toUpperCase();
      String vigenereEncrypted = vigenereEncrypt(plainText, key);
      String vigenereDecrypted = vigenereDecrypt(vigenereEncrypted, key);
      System.out.println("\nVigenere Cipher: "+vigenereEncrypted+"\nVigenere
Cipher Plain Text: "+vigenereDecrypted);
   }
   public static String vigenereEncrypt(String plainText, String key) {
      String encrypted = "";
      int n = plainText.length();
      int m = key.length();
      int count = 0;
      for (int i = 0; i < n; i++) {
         if(plainText.charAt(i) == ' ') {
            count++;
            continue;
         }
         int indexPT = alphabets.indexOf(plainText.charAt(i));
         int indexK = alphabets.indexOf(key.charAt((i-count)%m));
         encrypted +=
Character.toString(alphabets.charAt((indexPT+indexK)%26));
      }
      return encrypted;
   }
   public static String vigenereDecrypt(String encrypted, String key) {
      String decrypted = "";
      int n = encrypted.length();
      int m = key.length();
      for (int i = 0; i < n; i++) {
         int indexE = alphabets.indexOf(encrypted.charAt(i));
         int indexK = alphabets.indexOf(key.charAt(i%m));
```

```
            int indexDEC = indexE-indexK;
            if(indexDEC < 0)
                indexDEC += 26;
            decrypted += Character.toString(alphabets.charAt(indexDEC));
        }
        return decrypted;
    }
}
```

Output :



```
Enter plain text (only alphabets): vedansh
Enter vigenere cipher key (only alphabets): inslab

Vigenere Cipher: DRVLNTP
Vigenere Cipher Plain Text: VEDANSH

Process finished with exit code 0
```

Result: Thus the program to implement vigenere cipher encryption technique was developed and executed.

EXPERIMENT – 3(a): RAIL FENCE CIPHER

AIM: To implement a program for encrypting a plain text and decrypting a cipher text using Rail fence Cipher transposition technique.

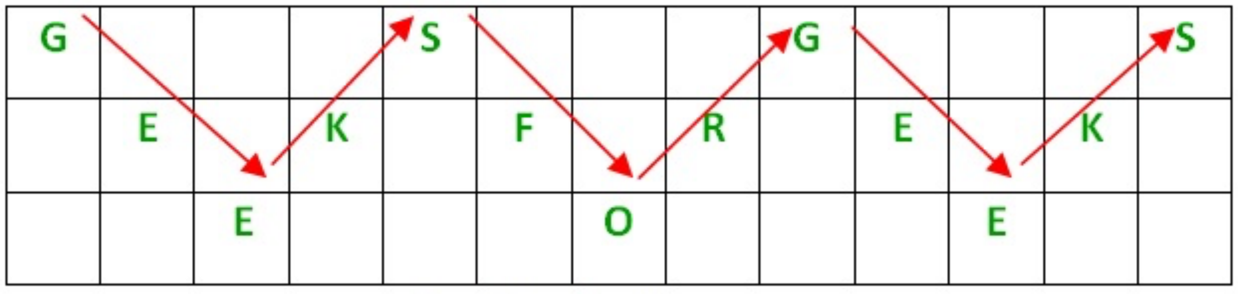
Requirements: C++ or any other language.

Theory: The rail fence cipher (also called a zigzag cipher) is a form of transposition cipher. It derives its name from the way in which it is encoded.

Encryption: In a transposition cipher, the order of the alphabets is re-arranged to obtain the cipher-text.

- In the rail fence cipher, the plain-text is written downwards and diagonally on successive rails of an imaginary fence.
- When we reach the bottom rail, we traverse upwards moving diagonally, after reaching the top rail, the direction is changed again. Thus the alphabets of the message are written in a zig-zag manner.
- After each alphabet has been written, the individual rows are combined to obtain the cipher-text.

For example, if the message is “GeeksforGeeks” and the number of rails = 3 then cipher is prepared as:



Input : "GeeksforGeeks "
Key = 3
Output : GsGsekfrek eoe

Decryption: As we've seen earlier, the number of columns in rail fence cipher remains equal to the length of plain-text message. And the key corresponds to the number of rails.

- Hence, rail matrix can be constructed accordingly. Once we've got the matrix we can figure-out the spots where texts should be placed (using the same way of moving diagonally up and down alternatively).
- Then, we fill the cipher-text row wise. After filling it, we traverse the matrix in zig-zag manner to obtain the original text.

Input : GsGsekfrek eoe
Key = 3
Output : "GeeksforGeeks "

Implementation:

Let cipher-text = "GsGsekfrek eoe" , and Key = 3

- Number of columns in matrix = len(cipher-text) = 13
- Number of rows = key = 3

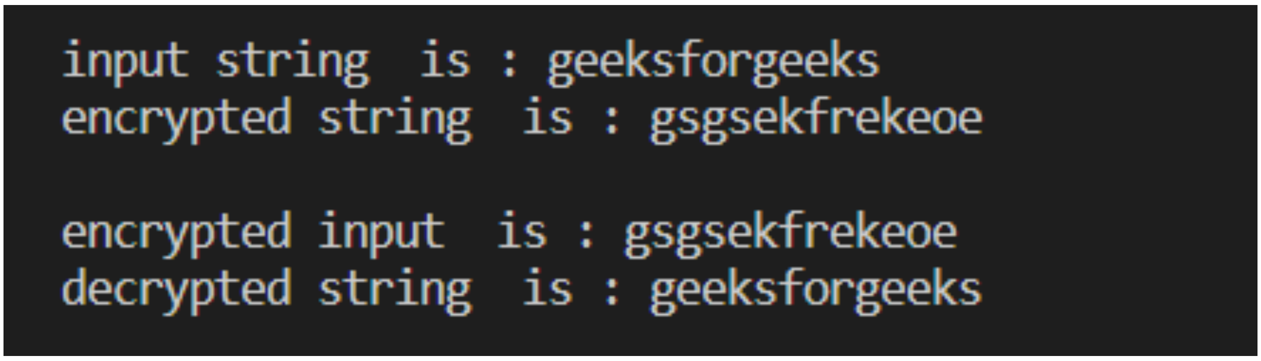
Hence original matrix will be of 3*13 , now marking places with text as '*' we get :

```
* _ _ _ * _ _ _ * _ _ _ *

_ * _ * _ * _ * _ * _ *

_ _ * _ _ _ *  _ _ _ * _
```

Program:

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
void encrypt_railfence(string input, int key) {
    int size = input.size();
    string cipher = "";
    char mat[key][size];
    for (int i = 0; i < key; i++)
        for (int j = 0; j < size; j++)
            mat[i][j] = '\n';
    int j = 0, dir = 1;
    for (int i = 0; i < size; i++) {
        mat[j][i] = input[i];
        j = j + dir;
        if (j == key - 1)
            dir = -1;
        else if (j == 0)
            dir = 1;
    }
    for (int k = 0; k < key; k++) {
        for (int i = 0; i < size; i++) {
            if (mat[k][i] != '\n')
                cipher += mat[k][i];
        }
    }
    cout << "\n input string  is : " << input;
    cout << "\n encrypted string  is : " << cipher;
}
void decrypt_railfence(string input, int key) {
    int size = input.size();
    string decipher = "";
    char mat[key][size];
    for (int i = 0; i < key; i++)
        for (int j = 0; j < size; j++)
            mat[i][j] = '\n';
    int pos = 0;
    for (int k = 0; k < key; k++) {
        int j = 0, dir = 1;
        for (int i = 0; i < size; i++) {
            if (j == k)
                mat[j][i] = input[pos++];
            j = j + dir;
            if (j == key - 1)
                dir = -1;
            else if (j == 0)
                dir = 1;
        }
    }
    int j = 0, dir = 1;
    for (int i = 0; i < size; i++) {
        decipher += mat[j][i];
        j = j + dir;
        if (j == key - 1)
            dir = -1;
        else if (j == 0)
            dir = 1;
    }
    cout << "\n encrypted input  is : " << input;
    cout << "\n decrypted string  is : " << decipher;
}
int main() {
    system("cls");
    encrypt_railfence("geeksforgeeks", 3);
    cout << "\n";
    decrypt_railfence("gsgsekfrekeoe", 3);
    cout << "\n\n";
    return 0;
}
```

Output :



```
input string  is : geeksforgeeks
encrypted string  is : gsgsekfrekeoe

encrypted input  is : gsgsekfrekeoe
decrypted string  is : geeksforgeeks
```

Result: Thus the program to implement rail fence cipher encryption technique was developed and executed.

EXPERIMENT – 3(b): COLUMNAR TRANSFORMATION CIPHER

AIM: To implement a program for encrypting a plain text and decrypting a cipher text using Columnar Transformation Cipher transposition technique.

Requirements: C++ or any other language.

Theory: The Columnar Transposition Cipher is a form of transposition cipher just like Rail Fence Cipher. Columnar Transposition involves writing the plaintext out in rows, and then reading the ciphertext off in columns one by one.

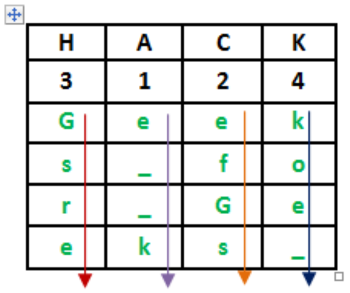
Encryption: In a transposition cipher, the order of the alphabets is re-arranged to obtain the cipher-text.

- The message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order.
- Width of the rows and the permutation of the columns are usually defined by a keyword.
- For example, the word HACK is of length 4 (so the rows are of length 4), and the permutation is defined by the alphabetical order of the letters in the keyword. In this case, the order would be “3 1 2 4”.
- Any spare spaces are filled with nulls or left blank or placed by a character (Example: _).
- Finally, the message is read off in columns, in the order specified by the keyword.

Encryption

Given text = Geeks for Geeks

Keyword = HACK **Length of Keyword** = 4 (no of rows) **Order of Alphabets in HACK** = 3124



H	A	C	K
3	1	2	4
G	e	e	k
s	_	f	o
r	_	G	e
e	k	s	_

Print Characters of column 1,2,3,4

Encrypted Text = e kefGsGsrekoe_

Input : Geeks for Geeks
Key = HACK
Output : e kefGsGsrekoe_

Decryption:

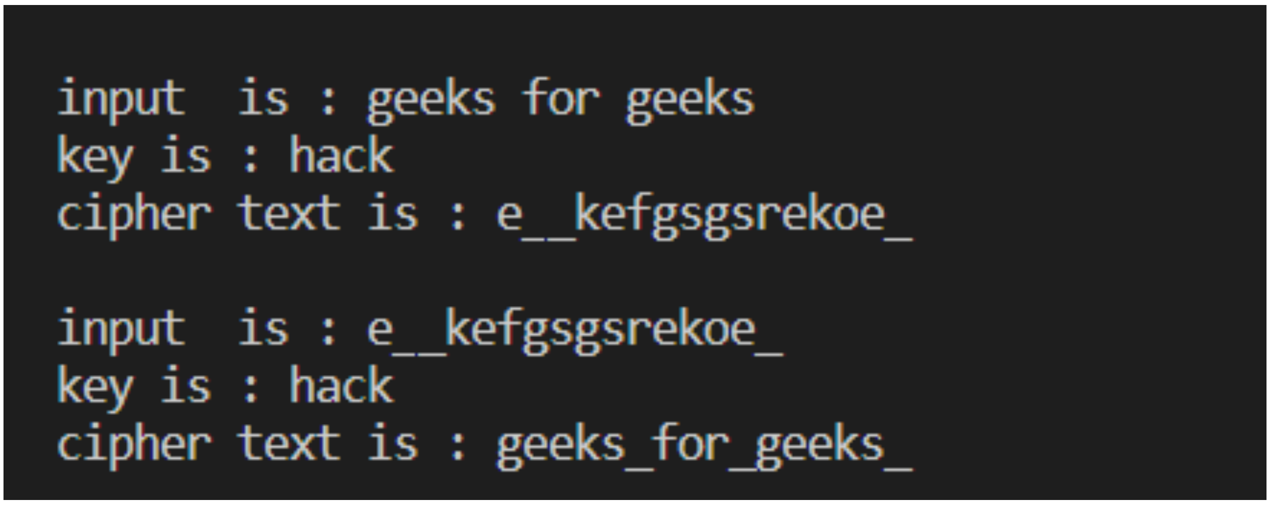
- To decipher it, the recipient has to work out the column lengths by dividing the message length by the key length.
- Then, write the message out in columns again, then re-order the columns by reforming the key word.

Input : e kefGsGsrekoe_
Key = HACK
Output : Geeks for Geeks

Program:

```
#include <iostream>
#include <math.h>
#include <string>
using namespace std;
string columnar(string input, string key, int task) {
    int keysize = key.size();
    float row = (input.size() * 1.0) / 4;
    int rows = ceil(row);
    string res = "";
    string key2 = key;
    int order[keysize];
    char mat[rows][keysize];
    for (int k = 0; k < keysize; k++) {
        char smallest = key[0];
        int pos = 0;
        for (int i = 0; i < keysize; i++) {
            if (key[i] < smallest) {
                smallest = key[i];
                pos = i;
            }
        }
        key[pos] = (char) 123;
        order[k] = pos;
    }
    if (task > 0)
    {
        int count = 0;
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < keysize; j++) {
                if (count >= input.size())
                    mat[i][j] = '_';
                else if (isspace(input[count])) {
                    mat[i][j] = '_';
                    count++;
                } else
                    mat[i][j] = input[count++];
            }
        }
        for (int i = 0; i < keysize; i++) {
            for (int j = 0; j < rows; j++) {
                res += mat[j][order[i]];
            }
        }
    } else
    {
        int count = 0;
        for (int i = 0; i < keysize; i++) {
            for (int j = 0; j < rows; j++) {
                mat[j][order[i]] = input[count++];
            }
        }
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < keysize; j++) {
                res += mat[i][j];
            }
        }
    }
    return res;
}
int main() {
    system("cls");
    string input = "geeks for geeks";
    string key = "hack";
    string cipher = columnar(input, key, 1);
    cout << "\n input  is : " << input;
    cout << "\n key is : " << key;
    cout << "\n cipher text is : " << cipher << endl;
    string decipher = columnar(cipher, key, -1);
    cout << "\n input  is : " << cipher;
    cout << "\n key is : " << key;
    cout << "\n cipher text is : " << decipher;
    cout << "\n\n";
    return 0;
}
```

Output :



```
input  is : geeks for geeks
key is : hack
cipher text is : e__kefgsgsrekoe_

input  is : e__kefgsgsrekoe_
key is : hack
cipher text is : geeks_for_geeks_
```

Result: Thus the program to implement columnar transformation cipher encryption technique was developed and executed.

EXPERIMENT – 4: Data Encryption Standard

AIM: To develop a program to implement Data Encryption Standard for encryption and decryption.

Requirements: C++ or any other language.

Theory: Simplified Data Encryption Standard (S-DES) is a simple version of the DES Algorithm. It is similar to the DES algorithm but is a smaller algorithm and has fewer parameters than DES. It was made for educational purposes so that understanding DES would become simpler. It is a block cipher that takes a block of plain text and converts it into ciphertext. It takes a block of 8 bits.

It is a symmetric key cipher i.e. they use the same key for both encryption and decryption. In this article, we are going to demonstrate key generation for s-des encryption and decryption algorithms. We take a random 10-bit key and produce two 8-bit keys which will be used for encryption and decryption.

Program:

```
#include <iostream>
#include <conio.h>
#include <algorithm>
using namespace std;
void printarray(int a[], int size) {
   for (int i = 0; i < size; i++)
      cout << a[i] << " ";
}

// utility function to convert binary to dec
int convert(int a) {
   int res = 0;
   switch (a) {
   case 0:
      res = 0;
      break;
   case 1:
      res = 1;
```

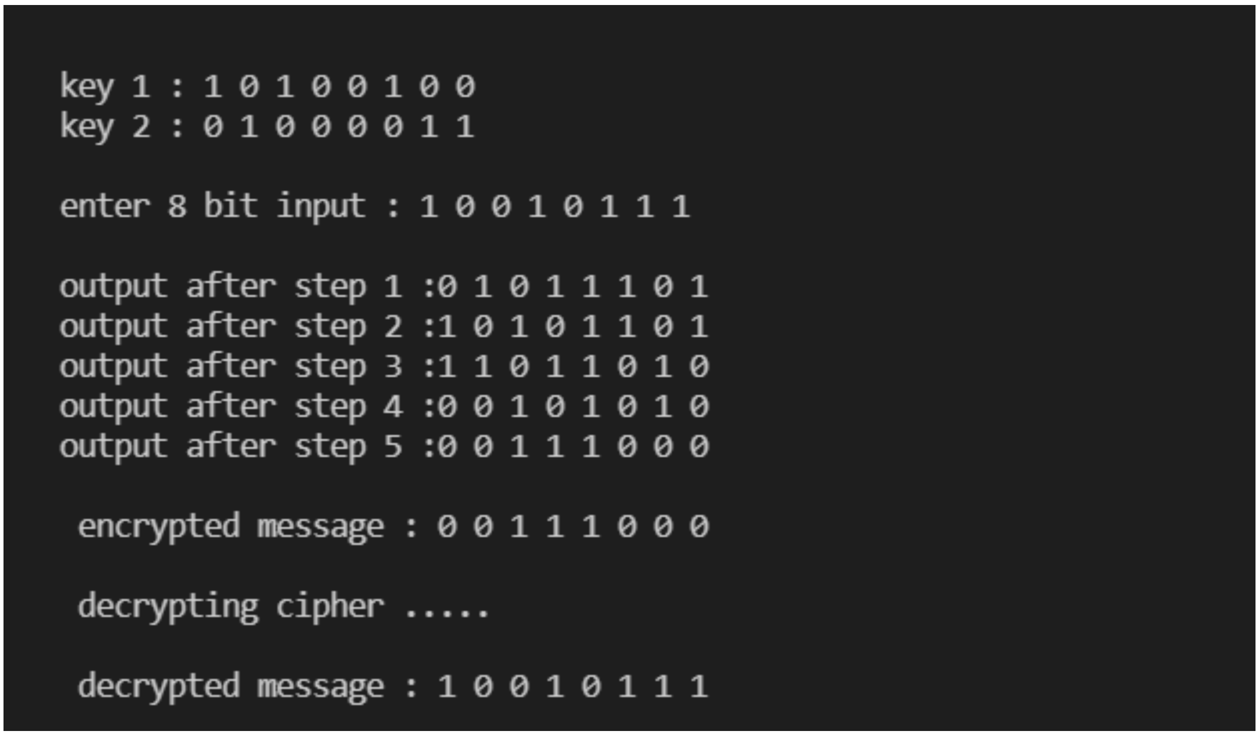
```
        break;
    case 10:
        res = 2;
        break;
    case 11:
        res = 3;
        break;
    }
    return res;
}

void init_permute(int arr[]) {
    int tmp[8];
    tmp[0] = arr[1];
    tmp[1] = arr[5];
    tmp[2] = arr[2];
    tmp[3] = arr[0];
    tmp[4] = arr[3];
    tmp[5] = arr[7];
    tmp[6] = arr[4];
    tmp[7] = arr[6];
    for (int i = 0; i < 8; i++)
        arr[i] = tmp[i];
}

void inverse_ip(int arr[]) {
    int tmp[8];
    tmp[0] = arr[3];
    tmp[1] = arr[0];
    tmp[2] = arr[2];
    tmp[3] = arr[4];
    tmp[4] = arr[6];
    tmp[5] = arr[1];
    tmp[6] = arr[7];
    tmp[7] = arr[5];
    for (int i = 0; i < 8; i++)
        arr[i] = tmp[i];
}
```

```
void p4(int arr[]) {
    int tmp[4];
    tmp[0] = arr[1];
    tmp[1] = arr[3];
    tmp[2] = arr[2];
    tmp[3] = arr[0];

    for (int i = 0; i < 4; i++)
        arr[i] = tmp[i];
}

void sbox(int s0[4][4], int s1[4][4], int arr[]) {
    int r1, c1, r2, c2;
    int tmp1, tmp2;
    // from s0
    r1 = arr[0] * 10 + arr[3];
    c1 = arr[1] * 10 + arr[2];
    r1 = convert(r1);
    c1 = convert(c1);
    tmp1 = s0[r1][c1];
    // from s1
    r2 = arr[4] * 10 + arr[7];
    c2 = arr[5] * 10 + arr[6];
    r2 = convert(r2);
    c2 = convert(c2);
    tmp2 = s1[r2][c2];

    switch (tmp1) {
    case 0:
        arr[0] = 0;
        arr[1] = 0;
        break;
    case 1:
        arr[0] = 0;
        arr[1] = 1;
        break;
    case 2:
        arr[0] = 1;
        arr[1] = 0;
        break;
    case 3:
        arr[0] = 1;
        arr[1] = 1;
        break;
    }
    switch (tmp2) {
    case 0:
        arr[2] = 0;
        arr[3] = 0;
        break;
    case 1:
        arr[2] = 0;
        arr[3] = 1;
        break;
    case 2:
        arr[2] = 1;
        arr[3] = 0;
        break;
    case 3:
        arr[2] = 1;
        arr[3] = 1;
        break;
    }
}

void exp_permute(int arr[]) {
    int tmp[8];

    tmp[0] = tmp[6] = arr[3];
    tmp[1] = tmp[7] = arr[0];
    tmp[2] = tmp[4] = arr[1];
    tmp[3] = tmp[5] = arr[2];

    for (int i = 0; i < 8; i++)
        arr[i] = tmp[i];
}
int main() {
    system("cls");
    int s0[4][4] = {
        {1, 0, 3, 2},
        {3, 2, 1, 0},
        {0, 2, 1, 3},
        {3, 1, 3,2}};

    int s1[4][4] = {
        {0, 1, 2, 3},
        {2, 0, 1, 3},
        {3, 0, 1, 0},
        {2, 1, 0, 3}};
    int key1[] = {1, 0, 1, 0, 0, 1, 0, 0}; // 8 bit keys
    int key2[] = {0, 1, 0, 0, 0, 0, 1, 1};
    cout << " \n key 1 : ";
    printarray(key1, 8);
    cout << "\n key 2 : ";
    printarray(key2, 8);
    // take 8 bit input
    int text[8] = {1, 0, 0, 1, 0, 1, 1, 1};
    cout << "\n\n enter 8 bit input : ";
    printarray(text, 8);
    //  cin >> text[0] >> text[1] >> text[2] >> text[3] >> text[4] >> text[5] >> text[6]
>> text[7];
    init_permute(text);
    cout << "\n\n output after step 1 :";
    printarray(text, 8);
    int l1[8], r1[8];
    int temp[8];
    copy_n(text, 4, l1);
    copy_n(text + 4, 4, r1);
    copy_n(r1, 8, temp);
    exp_permute(temp);
    for (int i = 0; i < 8; i++)
        temp[i] = temp[i] ^ key1[i];
    sbox(s0, s1, temp);
    p4(temp);
    for (int i = 0; i < 4; i++)
        temp[i] = temp[i] ^ l1[i];
    for (int i = 0; i < 4; i++) {
```

```
    text[i] = temp[i];
    text[i + 4] = r1[i];
}
cout << "\n output after step 2 :";
printarray(text, 8);
for (int i = 0; i < 4; i++) {
    int tmp = text[i];
    text[i] = text[i + 4];
    text[i + 4] = tmp;
}
cout << "\n output after step 3 :";
printarray(text, 8);
copy_n(text, 4, l1);
copy_n(text + 4, 4, r1);
copy_n(r1, 8, temp);
exp_permute(temp);
for (int i = 0; i < 8; i++)
    temp[i] = temp[i] ^ key2[i];
sbox(s0, s1, temp);
p4(temp);
for (int i = 0; i < 4; i++)
    temp[i] = temp[i] ^ l1[i];
for (int i = 0; i < 4; i++) {
    text[i] = temp[i];
    text[i + 4] = r1[i];
}
cout << "\n output after step 4 :";
printarray(text, 8);
inverse_ip(text);
cout << "\n output after step 5 :";
printarray(text, 8);
cout << "\n\n  encrypted message : ";
printarray(text, 8);
init_permute(text);
copy_n(text, 4, l1);
copy_n(text + 4, 4, r1);
copy_n(r1, 8, temp);
exp_permute(temp);
for (int i = 0; i < 8; i++)
```

```
        temp[i] = temp[i] ^ key2[i];
    sbox(s0, s1, temp);
    p4(temp);
    for (int i = 0; i < 4; i++)
        temp[i] = temp[i] ^ l1[i];
    for (int i = 0; i < 4; i++) {
        text[i] = temp[i];
        text[i + 4] = r1[i];
    }
    for (int i = 0; i < 4; i++) {
        int tmp = text[i];
        text[i] = text[i + 4];
        text[i + 4] = tmp;
    }
    copy_n(text, 4, l1);
    copy_n(text + 4, 4, r1);
    copy_n(r1, 8, temp);
    exp_permute(temp);
    for (int i = 0; i < 8; i++)
        temp[i] = temp[i] ^ key1[i];
    sbox(s0, s1, temp);
    p4(temp);
    for (int i = 0; i < 4; i++)
        temp[i] = temp[i] ^ l1[i];
    for (int i = 0; i < 4; i++) {
        text[i] = temp[i];
        text[i + 4] = r1[i];
    }
    inverse_ip(text);
    cout << "\n\n  decrypting cipher ….. \n\n  decrypted message : ";
    printarray(text, 8);
    cout << "\n\n";
    return 0;
}
```

Output :



```
key 1 : 1 0 1 0 0 1 0 0
key 2 : 0 1 0 0 0 0 1 1

enter 8 bit input : 1 0 0 1 0 1 1 1

output after step 1 :0 1 0 1 1 1 0 1
output after step 2 :1 0 1 0 1 1 0 1
output after step 3 :1 1 0 1 1 0 1 0
output after step 4 :0 0 1 0 1 0 1 0
output after step 5 :0 0 1 1 1 0 0 0

 encrypted message : 0 0 1 1 1 0 0 0

 decrypting cipher .....

 decrypted message : 1 0 0 1 0 1 1 1
```

Result: Thus the program to implement DES encryption technique was developed and executed.

EXPERIMENT – 5: Advanced Encryption Standard

AIM: To develop a program to implement Advanced Encryption Standard for encryption and decryption.

Requirements: C++ or any other language.

Theory: AES is based on a design principle known as a Substitution permutation network. It is fast in both software and hardware. Unlike its predecessor, DES, AES does not use a Feistel network. AES has a fixed block size of 128 bits and a key size of 128, 192, or 256 bits, whereas Rijndael can be specified with block and key sizes in any multiple of 32 bits, with a minimum of 128 bits. The block size has a maximum of 256 bits, but the key size has no theoretical maximum. AES operates on a 4×4 array of bytes, termed the state (versions of Rijndael with a larger block size have additional columns in the state). Most AES calculations are done in a special finite field. The AES cipher is specified as a number of repetitions of transformation rounds that convert the input plaintext into the final output of ciphertext. Each round consists of several processing steps, including one that depends on the encryption key. A set of reverse rounds are applied to transform ciphertext back into the original plaintext using the same encryption key.

Description of the algorithm

Key Expansion

Round keys are derived from the cipher key using Rijndael's key schedule

Initial Round

Add Round Key—each byte of the state is combined with the round key using bitwise xor

Rounds

SubBytes—a non-linear substitution step where each byte is replaced with another according to a lookup table.

ShiftRows—a transposition step where each row of the state is shifted cyclically a certain number of steps.

MixColumns—a mixing operation which operates on the columns of the state, combining the four bytes in each column.

AddRoundKey

Final Round

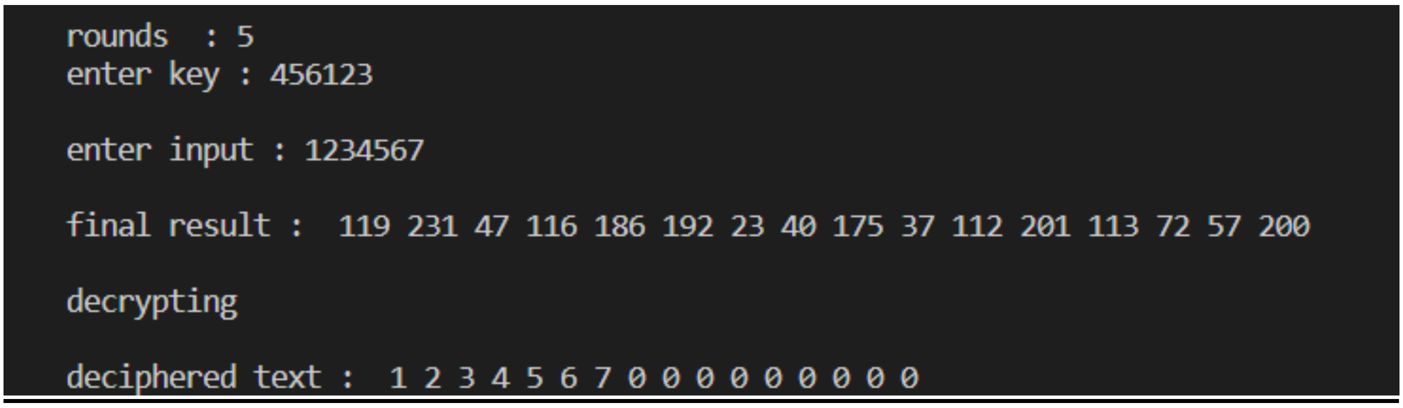
(no MixColumns)
SubBytes
ShiftRows
AddRoundKey

Program:

```
#include <iostream>
#include <bits/stdc++.h>
#include <math.h>
#include <vector>
#include <string>
using namespace std;
void print(vector < int > array) {
   for (int i = 0; i < array.size(); i++)
      cout << " " << array[i];
}
void s_box(vector < int > & input, vector < int > table) {
   for (int i = 0; i < input.size(); i++) {
      input[i] = table[input[i]];
   }
}
void addkey(vector < int > & input, vector < int > key) {
   for (int i = 0; i < input.size(); i++) {
      input[i] = input[i] ^ key[i];
   }
}
void shiftrow(vector < int > & input, int dir) {
   vector < vector < int >> mat(4, vector < int > (4, 0));
   for (int i = 0; i < 4; i++)
      for (int j = 0; j < 4; j++)
         mat[j][i] = input[4 * i + j];
   for (int i = 0; i < 4; i++) {
      for (int j = 0; j < i; j++) {
         if (dir > 0) {
            int tmp = mat[i][0];
            mat[i].erase(mat[i].begin());
            mat[i].push_back(tmp);
        } else {
            int tmp = mat[i][3];
            mat[i].erase(mat[i].begin() + 3);
            mat[i].insert(mat[i].begin(), tmp);
        }
    }
  }
  for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 4; j++) {
        input[4 * i + j] = mat[j][i];
    }
  }
}
int mul(int a_, int b_) {
  unsigned char a = (unsigned char) a_;
  unsigned char b = (unsigned char) b_;
  unsigned char p = 0;
  unsigned char counter;
  unsigned char hi_bit_set;
  for (counter = 0; counter < 8; counter++) {
    if ((b & 1) == 1)
        p ^= a;
    hi_bit_set = (a & 128);
    a <<= 1;
    if (hi_bit_set == 128)
        a ^= 27;
    b >>= 1;
  }
  int res = (int) p;
  return res;
}
void mixcol(vector < int > & input, int mixcol[4][4]) {
  int mat[4];
  for (int z = 0; z < 4; z++) {
    for (int i = 0; i < 4; i++)
        mat[i] = input[z * 4 + i];

    for (int i = 0; i < 4; i++) {
            int tmp = 0;
            for (int j = 0; j < 4; j++) {
                int tmp2 = mul(mat[j], mixcol[i][j]);
                tmp = tmp ^ tmp2;
            }
            input[z * 4 + i] = tmp;
        }
    }
}

void getroundkey(vector < int > & roundkey, vector < int > key, int n) {
    for (int i = 0; i < key.size(); i++)
        roundkey[i] = key[i];
    // left shift
    for (int i = 0; i < n; i++) {
        int tmp = roundkey[0];
        roundkey.erase(roundkey.begin());
        roundkey.push_back(tmp);
    }
}

int main() {
    system("cls");
    vector < int > input(16, 0);
    vector < int > key(16, 0);
    vector < int > sbox_table;
    vector < int > in_sbox_table(256, 0);

    // fill s-box table
    for (int i = 0; i < 256; i++)
        sbox_table.push_back(i);

    random_shuffle(sbox_table.begin(), sbox_table.end());

    for (int i = 0; i < 256; i++)
        in_sbox_table[sbox_table[i]] = i;

    int in_mixcol_mat[4][4] = {
        {14,11,13,9},
        {9,14,11,13},
        {13,9,14,11},
        {11,13,9,14}};

    int mixcol_mat[4][4] = {
        {2,3,1,1},
        {1,2,3,1},
        {1,1,2,3},
        {3,1,1,2}
    };

    int rounds = 5, roundcount = 1;
    string key_string = "", input_string = "";

    cout << " \n rounds  : " << rounds;
    cout << " \n enter key : ";
    cin >> key_string;
    cout << " \n enter input : ";
    cin >> input_string;

    for (int i = 0; i < key_string.size() && i < 16; i++)
        key[i] = stoi(key_string.substr(i, 1));
    for (int i = 0; i < input_string.size() && i < 16; i++)
        input[i] = stoi(input_string.substr(i, 1));

    // initial key add

    addkey(input, key);

    vector < int > roundkey(key);

    for (int i = 0; i < rounds - 1; i++) {
        s_box(input, sbox_table);
        shiftrow(input, 1);
        mixcol(input, mixcol_mat);
        getroundkey(roundkey, key, roundcount++);
        addkey(input, roundkey);
    }
    // final round
```

```
    s_box(input, sbox_table);
    shiftrow(input, 1);
    getroundkey(roundkey, key, roundcount);
    addkey(input, roundkey);

    cout << " \n final result : ";
    print(input);

    //  decryption part

    cout << "\n\n decrypting \n";
    addkey(input, roundkey);
    shiftrow(input, -1); // inverse shoft row
    s_box(input, in_sbox_table);

    for (int i = 0; i < rounds - 1; i++) {
        getroundkey(roundkey, key, --roundcount);
        addkey(input, roundkey);
        mixcol(input, in_mixcol_mat);
        shiftrow(input, -1);
        s_box(input, in_sbox_table);
    }
    // final key add
    addkey(input, key);
    cout << "\n deciphered text : ";
    print(input);
    return 0;
}
```

Output :



```
rounds   : 5
enter key : 456123

enter input : 1234567

final result :  119 231 47 116 186 192 23 40 175 37 112 201 113 72 57 200

decrypting

deciphered text :  1 2 3 4 5 6 7 0 0 0 0 0 0 0 0 0
```

Result: Thus the program to implement AES encryption technique was developed and executed.

EXPERIMENT – 6: RSA ALOGRITHM

AIM: To develop a program to implement RSA Algorithm.

Requirements: C++ or any other language.

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.

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.

Mechanism behind RSA algorithm :

>> Generating Public Key :

Select two prime no's. Suppose P = 53 and Q = 59.

Now First part of the Public key : n = P*Q = 3127.

We also need a small exponent say e :

But e Must be:

- An integer.
- Not be a factor of n.
- 1 < e < Φ(n) [Φ(n) is discussed below], Let us now consider it to be equal to 3.

Our Public Key is made of n and e

>> Generating Private Key :

We need to calculate Φ(n) :

Such that Φ(n) = (P-1)(Q-1)

so, Φ(n) = 3016

Now calculate Private Key, d :

d = (k*Φ(n) + 1) / e for some integer k

For k = 2, value of d is 2011.

Now we are ready with our – Public Key (n = 3127 and e = 3) and Private Key(d = 2011)

Now we will encrypt "HI" :

Convert letters to numbers : H = 8 and I = 9

Thus Encrypted Data c = 89e mod n.

Thus our Encrypted Data comes out to be 1394

Now we will decrypt 1394 :

Decrypted Data = cd mod n.

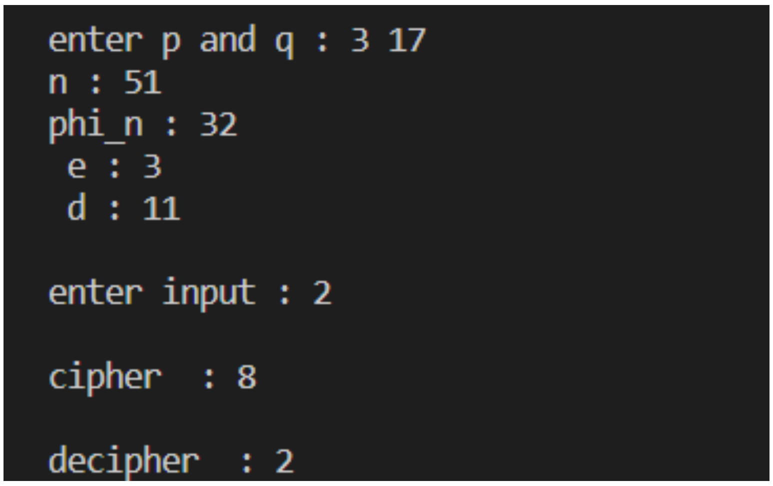
Thus our Encrypted Data comes out to be 89

8 = H and I = 9 i.e. "HI".

Program:

```
#include <iostream>
#include <cmath>
#include <algorithm>
#include <vector>
using namespace std;
long long int getpow(int a, int b) {
  if (b == 1)
    return a;
  long long int res = 1;
  for (int i = 1; i <= b; i++)
    res *= a;
  return res;
}
int main() {
    system("cls");
    int p = 3, q = 11, n, phi_n, e, d = 0;
    cout << "\n enter p and q : ";
    cin >> p >> q;
    n = p * q;
    cout << " n : " << n;
    phi_n = (p - 1) * (q - 1);
    cout << "\n phi_n : " << phi_n;
    vector < int > e_list;
    for (int i = 2; i < phi_n; i++) {
      int tmp = __gcd(phi_n, i);

      if (tmp == 1)
        e_list.push_back(i);
    }
    e = e_list[0];
    cout << "\n  e : " << e;
    for (int i = 1; i < phi_n; i++) {
      if ((i * e) % phi_n == 1) {
        d = i;
        break;
      }
    }
    cout << "\n  d : " << d;
    int input = 0;
    cout << "\n\n enter input : ";
    cin >> input;
    long long int cipher_ = getpow(input, e);
    long long int cipher = cipher_ % n;
    cout << "\n cipher  : " << cipher << endl;
    long long int decipher_ = getpow(cipher, d);
    long long int decipher = decipher_ % n;
    cout << "\n decipher  : " << decipher << endl;
    return 0;
}
```

Output :



```
enter p and q : 3 17
n : 51
phi_n : 32
 e : 3
 d : 11

enter input : 2

cipher  : 8

decipher  : 2
```

Result: Thus the program to implement RSA algorithm was developed and executed.

EXPERIMENT – 7: DHKE ALGORITHM

AIM: To implement Diffie-Hellman Key Exchange algorithm for a given problem.

Requirements: C++ or any other language.

Theory: Diffie–Hellman key exchange (D–H) is a specific method of exchanging keys. It is one of the earliest practical examples of Key exchange implemented within the field of cryptography. 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 communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher.

MECHANISM OF DHKE :

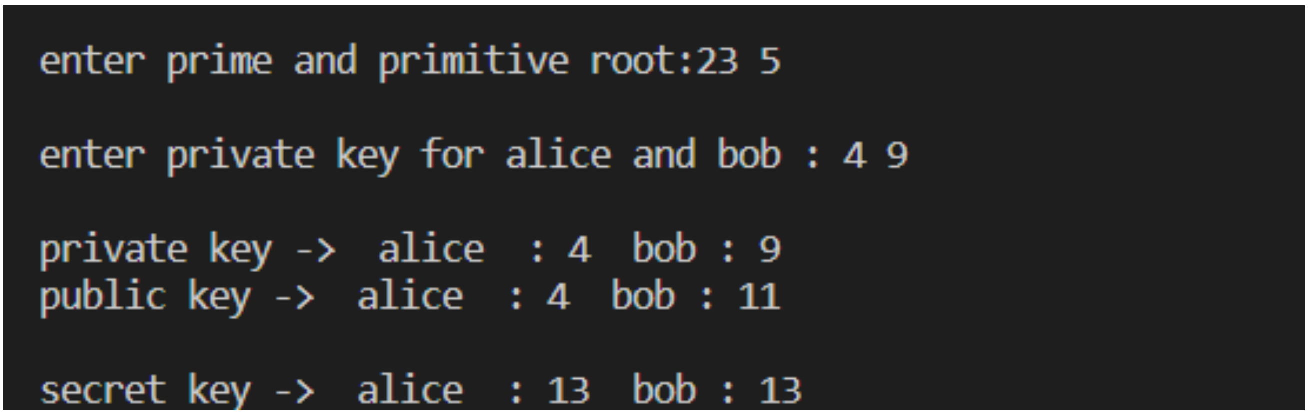
Alice				**Bob**		
Secret	Public	Calculus		Calculus	Public	Secret
	p, g				**p, g**	
a						**b**
		g^a mod p	$\rightarrow$		…	
	…		$\leftarrow$	g^b mod p		
$(g^b \bmod p)^a$ mod p			=			$(g^a \bmod p)^b$ mod p

1. Alice and Bob agree to use a prime number *p*=23 and base *g*=5.
2. Alice chooses a secret integer ***a***=**6**, then sends Bob A = *g**a*** mod *p*
    - A = 5**6** mod 23 = 8.
3. Bob chooses a secret integer ***b***=**15**, then sends Alice B = *g**b*** mod *p*
    - B = 5**15** mod 23 = 19.
4. Alice computes **s** = *B **a*** mod *p*
    - 19**6** mod 23 = **2**.
5. Bob computes **s** = *A **b*** mod *p*
    - 8**15** mod 23 = **2**.

Program:

```
#include <iostream>
#include <math.h>
#include <time.h>
#include <stdio.h>
using namespace std;
int getpow(int a, int b) {
   int res = 1;
   for (int i = 0; i < b; i++)
      res *= a;
   return res;
}
int main() {
   system("cls");
   int prime, primitive_root;
   cout << "\n enter prime and primitive root:";
   cin >> prime >> primitive_root;
   srand(time(0));
   int private_alice, private_bob, public_alice, public_bob;
   cout << "\n enter private key for alice and bob : ";
   cin >> private_alice >> private_bob;
   cout << "\n private key ->  alice  : " << private_alice << "  bob : " <<
private_bob;
   public_alice = getpow(primitive_root, private_alice) % prime;
   public_bob = getpow(primitive_root, private_bob) % prime;
   cout << "\n public key ->  alice  : " << public_alice << "  bob : " << public_bob;
   int secretkey_alice, secretkey_bob;
   secretkey_alice = getpow(public_bob, private_alice) % prime;
   secretkey_bob = getpow(public_alice, private_bob) % prime;
   cout << "\n\n secret key ->  alice  : " << secretkey_alice << "  bob : " <<
secretkey_bob;
   return 0;
}
```

Output :



```
enter prime and primitive root:23 5

enter private key for alice and bob : 4 9

private key ->  alice  : 4  bob : 9
public key ->  alice  : 4  bob : 11

secret key ->  alice  : 13  bob : 13
```

Result: Thus the program to implement DHKE algorithm was developed and executed.

EXPERIMENT – 8: SHA-1 ALGORITHM

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

Requirements: Python or any other language.

Theory: Secure Hash Algorithms, also known as SHA, are a family of cryptographic functions designed to keep data secured. It works by transforming the data using a hash function: an algorithm that consists of bitwise operations, modular additions, and compression functions. The hash function then produces a fixed-size string that looks nothing like the original. These algorithms are designed to be one-way functions, meaning that once they're transformed into their respective hash values, it's virtually impossible to transform them back into the original data. A few algorithms of interest are SHA-1, SHA-2, and SHA-3, each of which was successively designed with increasingly stronger encryption in response to hacker attacks. SHA-0, for instance, is now obsolete due to the widely exposed vulnerabilities.

A common application of SHA is to encrypting passwords, as the server side only needs to keep track of a specific user's hash value, rather than the actual password. This is helpful in case an attacker hacks the database, as they will only find the hashed functions and not the actual passwords, so if they were to input the hashed value as a password, the hash function will convert it into another string and subsequently deny access. Additionally, SHAs exhibit the avalanche effect, where the modification of very few letters being encrypted causes a big change in output; or conversely, drastically different strings produce similar hash values. This effect causes hash values to not give any information regarding the input string, such as its original length. In addition, SHAs are also used to detect the tampering of data by attackers, where if a text file is slightly changed and barely noticeable, the modified file's hash value will be different than the original file's hash value, and the tampering will be rather noticeable.

Pseudocode

Suppose the message 'abc' were to be encoded using SHA-1, with the message 'abc' in binary being:

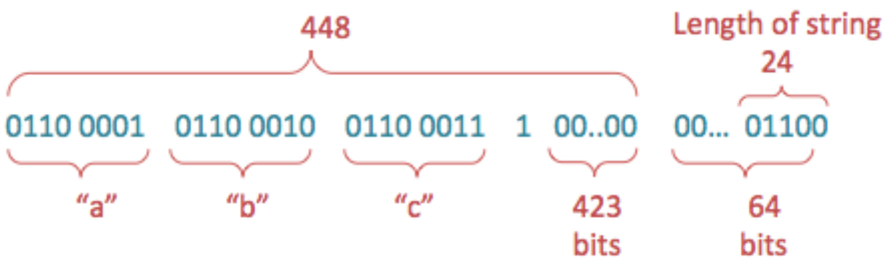
01100001\ 01100010\ 0110001101100001 01100010 01100011 and that in hex being:

616263.616263.

1) The first step is to initialize five random strings of hex characters that will serve as part of the hash function (shown in hex):

$$
\begin{aligned}
H_0 &= 67DE2A01\\
H_1 &= BB03E28C\\
H_2 &= 011EF1DC\\
H_3 &= 9293E9E2\\
H_4 &= CDEF23A9.
\end{aligned}
$$

2) The message is then padded by appending a 1, followed by enough 0s until the message is 448 bits. The length of the message represented by 64 bits is then added to the end, producing a message that is 512 bits long:



Padding of string "abc" in bits, finalized by the length of the string, which is 24 bits.

3) The padded input obtained above, MM, is then divided into 512-bit chunks, and each chunk is further divided into sixteen 32-bit words, W_0 … W_{15}W0…W15. In the case of 'abc', there's only one chunk, as the message is less than 512-bits total.

4) For each chunk, begin the 80 iterations, ii, necessary for hashing (80 is the determined number for SHA-1), and execute the following steps on each chunk, M_n:Mn:

• For iterations 16 through 79, where 16 \leq i \leq 7916≤i≤79, perform the following operation:

$$W(i) = S^1\big(W(i-3) \oplus W(i-8) \oplus W(i-14) \oplus W(i-16)\big),$$

where XOR, or \oplus⊕, is represented by the following comparison of inputs xx and y:y:

xx	yy	Output
0	0	0
1	0	1
0	1	1
1	1	0

For example, when ii is 16, the words chosen are W(13), W(8), W(2), W(0)W(13),W(8),W(2),W(0), and the output is a new word, W(16)W(16), so performing the XOR, or \oplus⊕, operation on those words will give this:

W(0)	01100001 01100010 01100011 10000000
W(2)	00000000 00000000 00000000 00000000
W(8)	00000000 00000000 00000000 00000000
W(13)	00000000 00000000 00000000 00000000
	⊕
W(16)	01100001 01100010 01100011 10000000

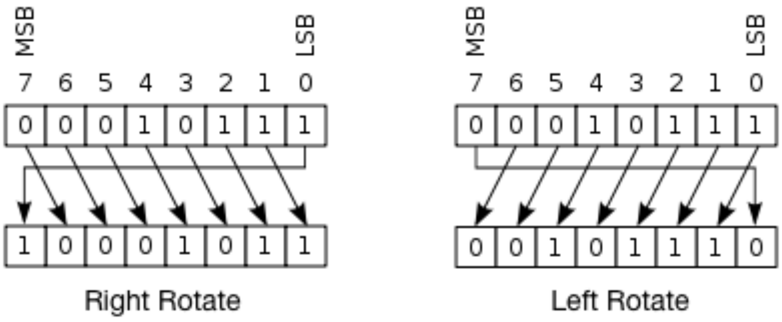
Circular Shift Operation

Now, the circular shift operation S^n(X)Sn(X) on the word XX by nn bits, nn being an integer between 00 and 3232, is defined by:

$$S^n(X) = (X << n) \quad \textbf{OR} \quad (X >> 32 - n),$$

where X<<nX<<n is the left-shift operation, obtained by discarding the leftmost nn bits of XX and padding the result with nn zeroes on the right.

X>> 32-nX>>32−n is the right-shift operation obtained by discarding the rightmost nn bits of XX and padding the result with nn zeroes on the left. Thus S^n(X)Sn(X) is equivalent to a circular shift of XX by nn positions, and in this case the circular left-shift is used.



So, a left shift S^n\big(W(i)\big),Sn(W(i)), where W(i)W(i) is 10010,10010, would produce 0100101001, as the rightmost bit 00 is shifted to the left side of the string. Therefore, W(16)W(16) would end up being

11000010 11000100 11000111 000000000.

5) Now, store the hash values defined in step 1 in the following variables:

$$
\begin{aligned}
A &= H_0 \\
B &= H_1 \\
C &= H_2 \\
D &= H_3 \\
E &= H_4.
\end{aligned}
$$

6) For 8080 iterations, where 0 \leq i \leq 790≤i≤79, compute

TEMP = S^5 *(A) + f(i; B, C, D) + E + W(i) + K(i).TEMP=S5∗(A)+f(i;B,C,D)+E+W(i)+K(i).

See below for details on the logical function, ff, and on the values of K(i).K(i).

Reassign the following variables:

$$\begin{aligned} E &= D \\ D &= C \\ C &= S^{30}(B) \\ B &= A \\ A &= TEMP. \end{aligned}$$

7) Store the result of the chunk's hash to the overall hash value of all chunks, as shown below, and proceed to execute the next chunk:

$$\begin{aligned} H_0 &= H_0 + A \\ H_1 &= H_1 + B \\ H_2 &= H_2 + C \\ H_3 &= H_3 + D \\ H_4 &= H_4 + E. \end{aligned}$$

8) As a final step, when all the chunks have been processed, the message digest is represented as the 160-bit string comprised of the OR logical operator, \lorV, of the 5 hashed values:

$$HH = S^{128}(H_0) \vee S^{96}(H_1) \vee S^{64}(H_2) \vee S^{32}(H_3) \vee H_4.$$

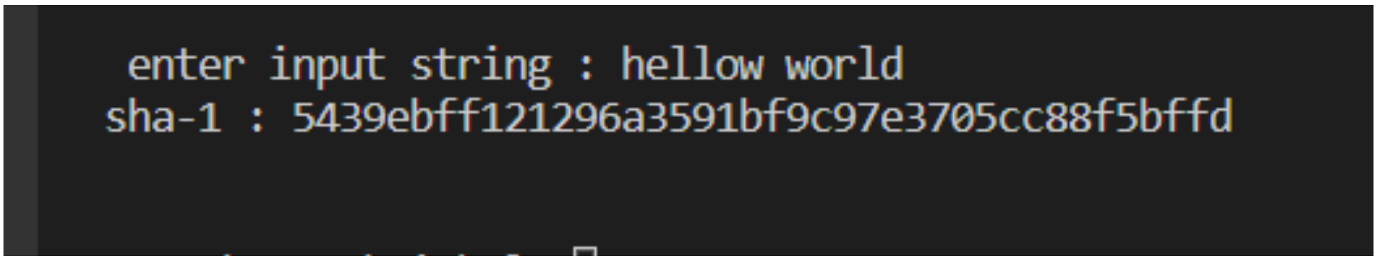
So, the string 'abc' becomes represented by a hash value akin to a9993e364706816aba3e25717850c26c9cd0d89d.

If the string changed to 'abcd', for instance, the hashed value would be drastically different so attackers cannot tell that it is similar to the original message. The hash value for 'abcd' is 81fe8bfe87576c3ecb22426f8e57847382917acf.

Program:

```
import hashlib
inputtext = input("\n enter input string : ")
hash = hashlib.sha1(inputtext.encode())
print("sha-1 : "+str(hash.hexdigest()))
print("\n")
```

Output :



```
 enter input string : hellow world
sha-1 : 5439ebff121296a3591bf9c97e3705cc88f5bffd
```

Result: Thus the program to implement SHA-1 algorithm was developed and executed.

EXPERIMENT – 9: DIGITAL SIGNATURE STANDARD

AIM: Implement the signature scheme: Digital Signature Standard.

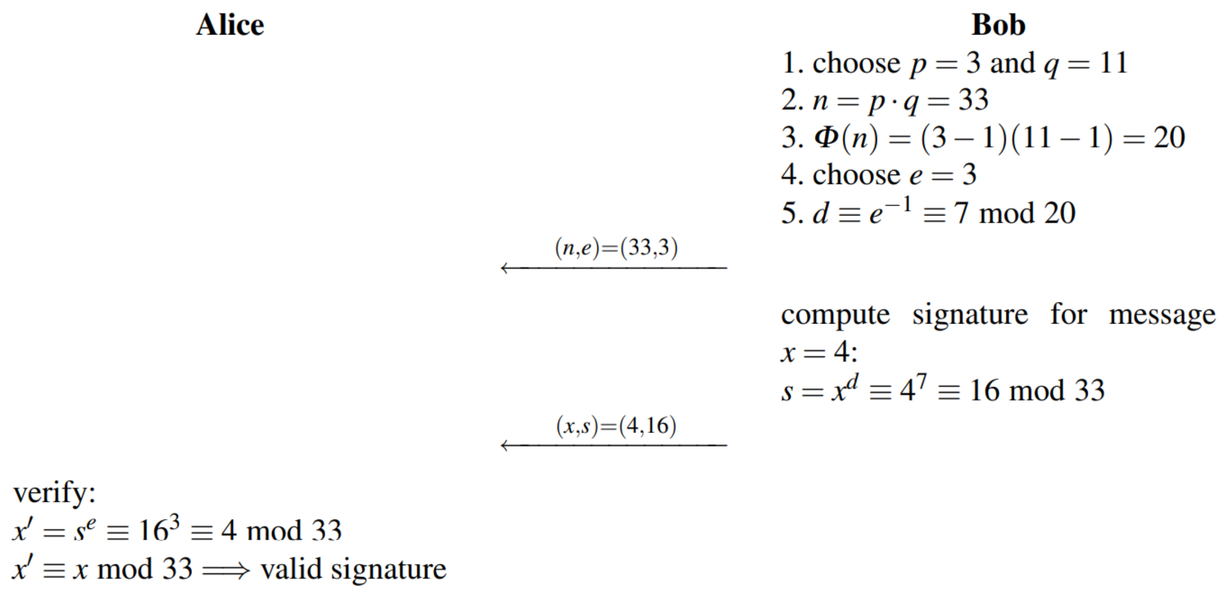
Requirements: C++ or any other language.

Theory:

Digital Signature : As the name sounds are the new alternative to sign a document digitally. It ensures that the message is sent by the intended user without any tampering by any third party (attacker). In simple words, digital signatures are used to verify the authenticity of the message sent electronically.

RSA: It is the most popular asymmetric cryptographic algorithm. It is primarily used for encrypting message s but can also be used for performing digital signature over a message.

Algorithm : Suppose Bob wants to send a signed message (x = 4) to Alice. The first steps are exactly the same as it is done for an RSA encryption: Bob computes his RSA parameters and sends the public key to Alice. In contrast to the encryption scheme, now the private key is used for signing while the public key is needed to verify the signature.

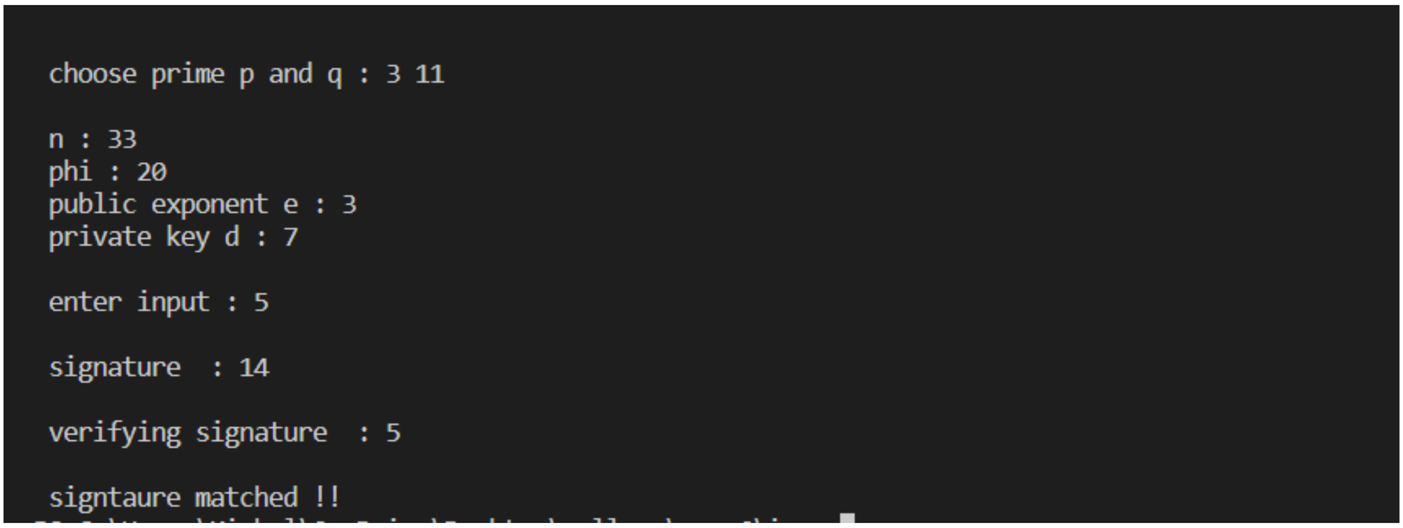

Alice
Bob
1. choose $p = 3$ and $q = 11$
2. $n = p \cdot q = 33$
3. $\Phi(n) = (3-1)(11-1) = 20$
4. choose $e = 3$
5. $d \equiv e^{-1} \equiv 7 \bmod 20$
$\xleftarrow{(n,e)=(33,3)}$
compute signature for message $x = 4$:
$s = x^d \equiv 4^7 \equiv 16 \bmod 33$
$\xleftarrow{(x,s)=(4,16)}$
verify:
$x' = s^e \equiv 16^3 \equiv 4 \bmod 33$
$x' \equiv x \bmod 33 \Longrightarrow$ valid signature

Program:

```
#include <iostream>
#include <algorithm>
using namespace std;
long long int getpow(int a, int b) {
   if (b == 1)
      return a;
   long long int res = 1;
   for (int i = 1; i <= b; i++)
      res *= a;
   return res;
}
int main() {
   system("cls");
   int p, q, n, phi_n = 0, e, d, x, s;
   cout << "\n choose prime p and q : ";
   cin >> p >> q;
   n = p * q;
   cout << "\n n : " << n;
   phi_n = (p - 1) * (q - 1);
   cout << "\n phi : " << phi_n;
   for (int i = 2; i < phi_n; i++) {
      if (__gcd(i, phi_n) == 1) {
         e = i;
         break;
      }
   }
   cout << "\n public exponent e : " << e;
   for (int i = 1; i < phi_n; i++) {
      if ((i * e) % phi_n == 1) {
         d = i;
         break;
      }
   }
   cout << "\n private key d : " << d;
   int input = 0;
   cout << "\n\n enter input : ";
   cin >> input;
   long long int sig = getpow(input, d);
```

```
    sig = sig % n;
    cout << "\n signature  : " << sig << endl;
    long long int sig_v = getpow(sig, e);
    sig_v = sig_v % n;
    cout << "\n verifying signature  : " << sig_v << endl;
    if (sig_v == input)
        cout << " \n signtaure matched !!";
    return 0;
}
```

Output :



```
choose prime p and q : 3 11

n : 33
phi : 20
public exponent e : 3
private key d : 7

enter input : 5

signature  : 14

verifying signature  : 5

signtaure matched !!
```

Result: Thus the program to implement Digital Signature Standard was developed and executed.

EXPERIMENT – 10: INTRUSION DETECTION SYSTEM

AIM: Demonstrate Intrusion Detection System using Snort tool.

Requirements: Snort IDS and npcap.

Theory: SNORT is a network based intrusion detection system which is written in C programming language. It was developed in 1998 by Martin Roesch. Now it is developed by Cisco. It is free open-source software. It can also be used as a packet sniffer to monitor the system in real time. The network admin can use it to watch all the incoming packets and find the ones which are dangerous to the system. It is based on library packet capture tool. The rules are fairly easy to create and implement and it can be deployed in any kind on operating system and any kind of network environment. The main reason of the popularity of this IDS over others is that it is a free-to-use software and also open source because of which any user can able to use it as the way he want.

Features:

- Real-time traffic monitor
- Packet logging
- Analysis of protocol
- Content matching
- OS fingerprinting
- Can be installed in any network environment.
- Creates logs
- Open Source
- Rules are easy to implement

Installation Steps:

In Windows:

- Step-1: Download SNORT installer from https://www.snort.org/downloads/snort/Snort_2_9_15_Installer.exe
- Step-2: Execute the Snort_2_9_15_Installer.exe
- Step-3: install npcap from https://npcap.com/#download
- Step-4:download snort rules from https://www.snort.org/downloads
- Step-5: put snort rules into rules folder inside snort directory and replace snort.config from etc folder with the one from downloaded snort ruled

Basic Usages:

- Sniffer Mode –

  To print TCP/IP header use command ./snort -v

  To print IP address along with header use command ./snort -vd

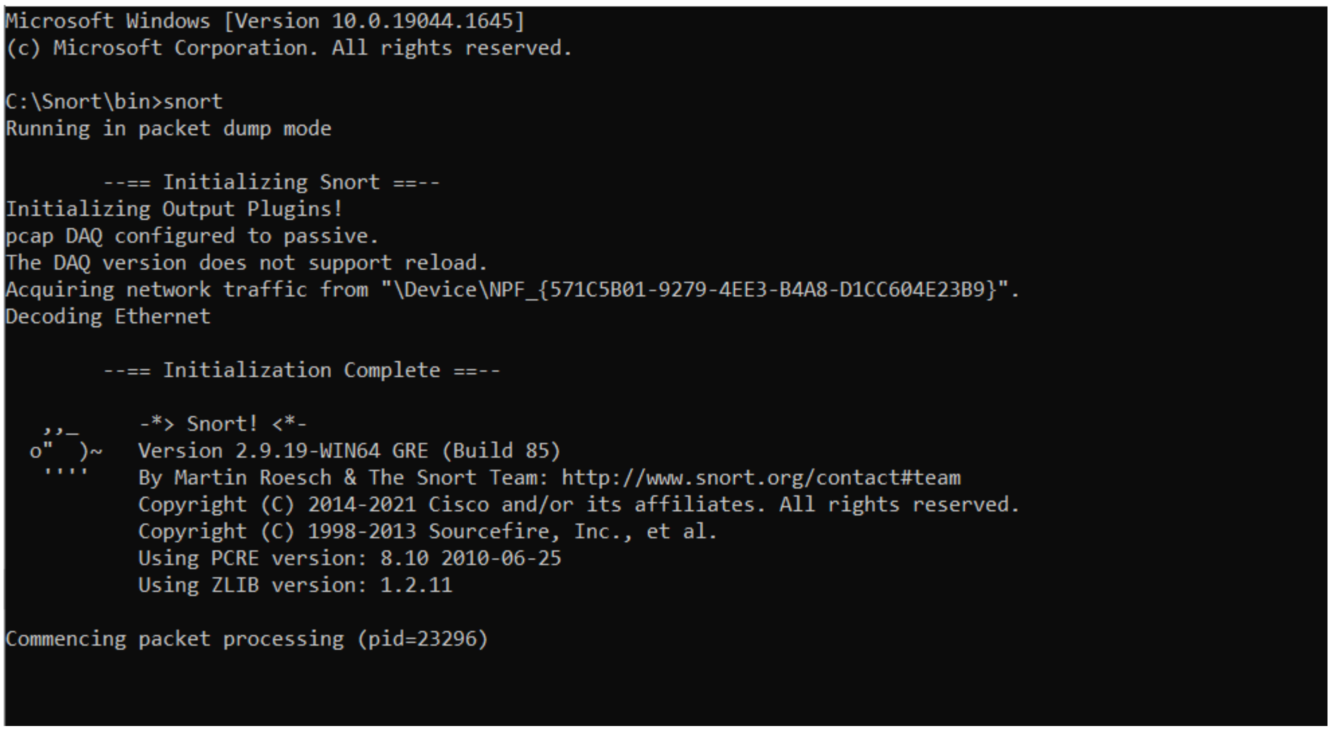
- Packet Logging –

  To store packet in disk you need to give path where you want to store the logs. For this command is./snort -dev -l ./SnortLogs.

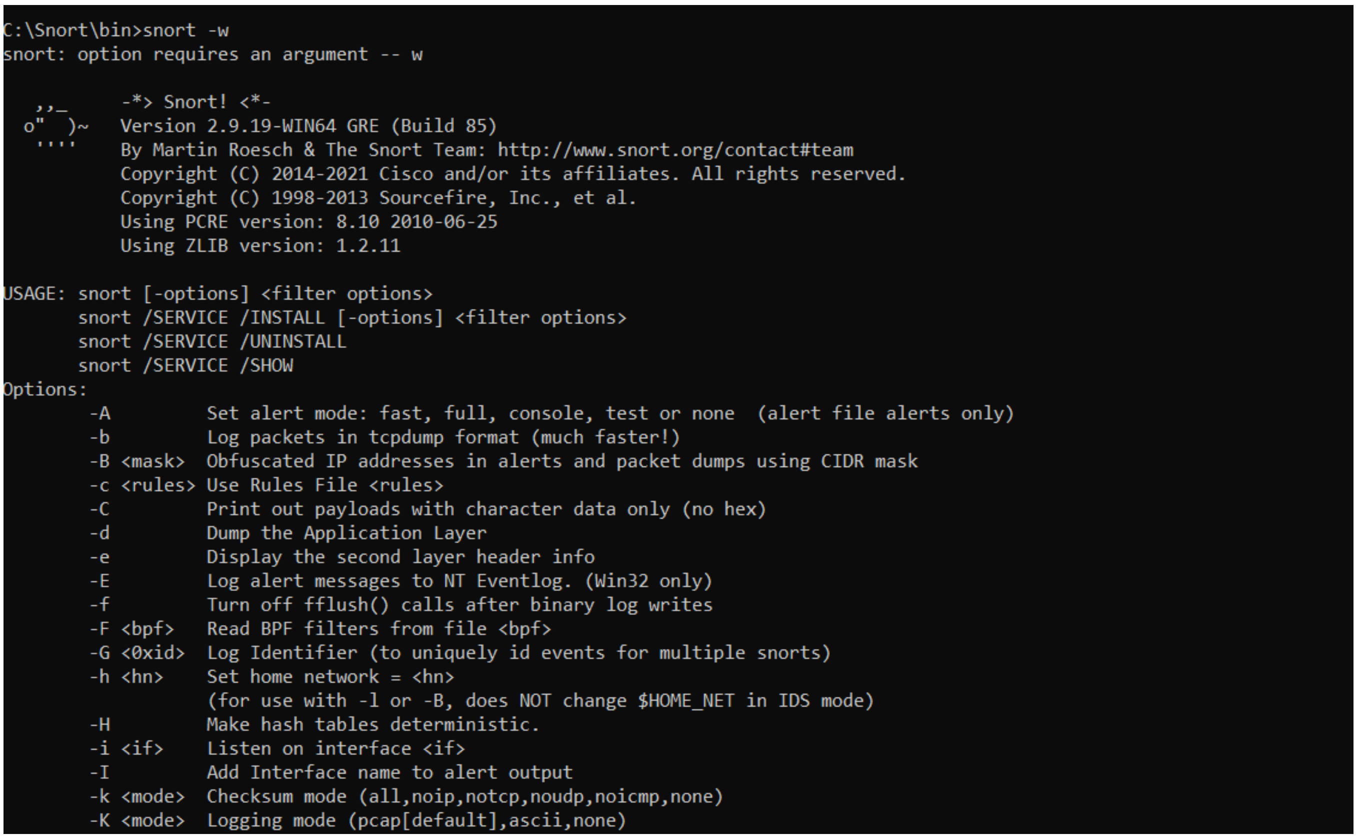
- Activate network intrusion detection mode –

  To start this mode use this command ./snort -dev -l ./SnortLogs -h 192.127.1.0/24 -c snort.conf

Implementation:



```
Microsoft Windows [Version 10.0.19044.1645]
(c) Microsoft Corporation. All rights reserved.

C:\Snort\bin>snort
Running in packet dump mode

        --== Initializing Snort ==--
Initializing Output Plugins!
pcap DAQ configured to passive.
The DAQ version does not support reload.
Acquiring network traffic from "\Device\NPF_{571C5B01-9279-4EE3-B4A8-D1CC604E23B9}".
Decoding Ethernet

        --== Initialization Complete ==--

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.19-WIN64 GRE (Build 85)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
           Copyright (C) 2014-2021 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using PCRE version: 8.10 2010-06-25
           Using ZLIB version: 1.2.11

Commencing packet processing (pid=23296)
```



```
C:\Snort\bin>snort -w
snort: option requires an argument -- w

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.19-WIN64 GRE (Build 85)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
           Copyright (C) 2014-2021 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using PCRE version: 8.10 2010-06-25
           Using ZLIB version: 1.2.11

USAGE: snort [-options] <filter options>
       snort /SERVICE /INSTALL [-options] <filter options>
       snort /SERVICE /UNINSTALL
       snort /SERVICE /SHOW
Options:
        -A         Set alert mode: fast, full, console, test or none  (alert file alerts only)
        -b         Log packets in tcpdump format (much faster!)
        -B <mask>  Obfuscated IP addresses in alerts and packet dumps using CIDR mask
        -c <rules> Use Rules File <rules>
        -C         Print out payloads with character data only (no hex)
        -d         Dump the Application Layer
        -e         Display the second layer header info
        -E         Log alert messages to NT Eventlog. (Win32 only)
        -f         Turn off fflush() calls after binary log writes
        -F <bpf>   Read BPF filters from file <bpf>
        -G <0xid>  Log Identifier (to uniquely id events for multiple snorts)
        -h <hn>    Set home network = <hn>
                   (for use with -l or -B, does NOT change $HOME_NET in IDS mode)
        -H         Make hash tables deterministic.
        -i <if>    Listen on interface <if>
        -I         Add Interface name to alert output
        -k <mode>  Checksum mode (all,noip,notcp,noudp,noicmp,none)
        -K <mode>  Logging mode (pcap[default],ascii,none)
```

Result: Thus intrusion detection system was implemented using snort tool.