Experiment : 1

Aim : Implement Caesar cipher encryption-decryption.

Program (Pract1.c) :

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

void encryption(char\*,int);

void decryption(char\*,int);

char PT[50],CT[50];

int main(){

    int k;

        /\*.................Encryption................\*/

    printf("\n...Encryption process...\n");

    printf("Enter Plain text : ");

    scanf("%s",PT);

    printf("Enter Key : ");

    scanf("%d",&k);

    encryption(PT,k);

    printf("Encrypted text is : %s\n\n",CT);

        /\*.................Decryption................\*/

    printf("\n...Decryption process...\n");

    printf("Enter Cipher text : ");

    scanf("%s",CT);

    printf("Enter Key : ");

    scanf("%d",&k);

    decryption(CT,k);

    printf("Decrypted text is : %s\n\n",PT);

    return 0;

}

void encryption(char\* str,int k){

    int l,i;

    l=strlen(str);

    for(i=0; i<l; i++){

        if(isalnum(str[i])){   // we are olny considering alphabets and putting other as it is

            CT[i]=str[i]+k;

        }

        else{

            CT[i]=str[i];

        }

    }

}

void decryption(char\* str,int k){

    int l,i;

    l=strlen(str);

    for(i=0; i<l; i++){         // we are olny considering alphabets and putting other as it is

        if(isalnum(str[i])){

            PT[i]=str[i]-k;

        }

        else{

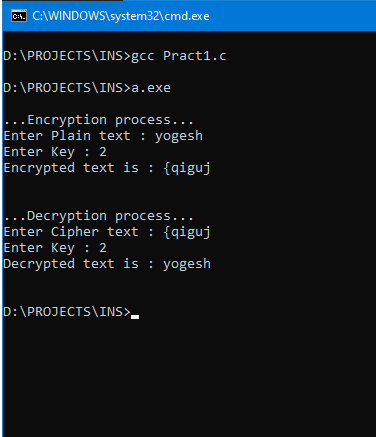
            PT[i]=str[i];

        }

    }

}

Output :



Experiment : 2

Aim : Implement Monoalphabetic cipher encryption-decryption.

Program (Pract2.c):

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#include <conio.h>

void encryption(char\*,char\*);

void decryption(char\*,char\*);

char PT[50],CT[50],K[26];

int main(){

    int i;

    /\*.................Encryption................\*/

    printf("\n...Encryption process...\n");

    printf("Enter Plain text : ");

    scanf("%s",PT);

    printf("\nEnter mapping for a to z : \n");

    printf("abcdefghijklmnopqrstuvwxyz\n");

    scanf("%s",K);

    printf("\n");

    encryption(PT,K);

    printf("Encrypted text is : %s\n\n\n",CT);

    /\*.................Decryption................\*/

    printf("\n...Decryption process...\n");

    printf("Enter Cipher text : ");

    scanf("%s",CT);

    printf("\nEnter mapping for a to z : \n");

    printf("abcdefghijklmnopqrstuvwxyz\n");

    scanf("%s",K);

    printf("\n");

    decryption(CT,K);

    printf("Decrypted text is : %s\n\n",PT);

    return 0;

}

void encryption(char\* str,char\* k){

    int i,l,c;

    l=strlen(str);

    for(i=0;i<l;i++){

        c=PT[i]-97;

        CT[i]=k[c];

    }

}

void decryption(char\* str,char\* k){

    int i,l,c;

    l=strlen(str);

    for(i=0;i<l;i++){

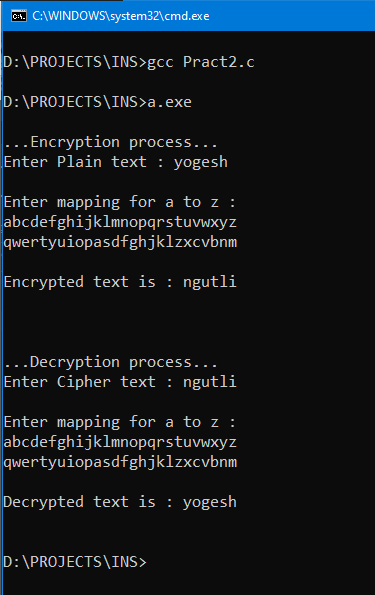
        c=PT[i]-97;

        CT[i]=k[c];

    }

}

Output :



Experiment : 3

Aim : Implement Playfair cipher encryption-decryption.  
  
Program (Pract3.c) :

#include <stdio.h>

#include <conio.h>

#include <string.h>

#include <ctype.h>

#include <stdlib.h>

#define SIZE 50

#define N 5

void toLowerCase(char [SIZE], int );

int removeSpaces(char\* , int );

void generateKeyTable(char[SIZE] , int  , char[N][N] );

void search(char[N][N], char , char , int[N] );

int prepare(char[SIZE], int );

void encrypt(char[SIZE], char [N][N], int );

void encryptByPlayfairCipher(char[SIZE], char[SIZE]);

int mod5(int );

void decryptByPlayfairCipher(char str[], char key[]);

void decrypt(char str[SIZE], char keyT[N][N], int ps);

int main(){

    char str[SIZE],key[SIZE];

        /\*.................Encryption................\*/

    printf("Playfair Cipher Encryption...\n");

    printf("Enter Plaintext : ");

    scanf("%s",str);

    printf("Enter Key : ");

    scanf("%s",key);

    encryptByPlayfairCipher(str, key);

    printf("Encrypted text : %s", str);

        /\*.................Decryption................\*/

    printf("\n\nPlayfair Cipher Decryption...\n");

    printf("Enter Ciphertext : ");

    scanf("%s",str);

    printf("Enter Key : ");

    scanf("%s",key);

    decryptByPlayfairCipher(str, key);

    printf("Decrypted text : %s", str);

 return 0;

}

void toLowerCase(char plain[], int ps){

    int i;

    for (i = 0; i < ps; i++) {

        if (plain[i] > 64 && plain[i] < 91)

        plain[i] += 32;

    }

}

int removeSpaces(char\* plain, int ps){

    int i, count = 0;

    for (i = 0; i < ps; i++)

        if (plain[i] != ' ')

            plain[count++] = plain[i];

    plain[count] = '\0';

    return count;

}

void generateKeyTable(char key[], int ks, char keyT[N][N]){

    int i, j, k, flag = 0, \*dicty;

    dicty = (int\*)calloc(26, sizeof(int));

    for (i = 0; i < ks; i++) {

        if (key[i] != 'j')       //exclude j

        dicty[key[i] - 97] = 2;

    }

    dicty['j' - 97] = 1;

    i = 0;

    j = 0;

    for (k = 0; k < ks; k++) {

        if (dicty[key[k] - 97] == 2) {

            dicty[key[k] - 97] -= 1;

            keyT[i][j] = key[k];

            j++;

            if (j == 5) {

                i++;

                j = 0;

            }

        }

    }

    for (k = 0; k < 26; k++) {

        if (dicty[k] == 0) {

            keyT[i][j] = (char)(k + 97);

            j++;

            if (j == 5) {

                i++;

                j = 0;

            }

        }

    }

}

void search(char keyT[N][N], char a, char b, int arr[]){

    int i, j;

    if (a == 'j')

    a = 'i';

    else if (b == 'j')

    b = 'i';

    for (i = 0; i < N; i++) {

        for (j = 0; j < N; j++) {

            if (keyT[i][j] == a) {

                arr[0] = i;

                arr[1] = j;

            }

            else if (keyT[i][j] == b) {

                arr[2] = i;

                arr[3] = j;

            }

        }

    }

}

int prepare(char str[], int ptrs){

    if (ptrs % 2 != 0) {

        str[ptrs++] = 'z';

        str[ptrs] = '\0';

    }

    return ptrs;

}

// void encrypt(char str[], char keyT[N][N], int ps){

void encrypt(char str[], char keyT[N][N], int ps){

    int i, a[4];

    for (i = 0; i < ps; i += 2) {

        search(keyT, str[i], str[i + 1], a);

        if (a[0] == a[2]) {

            str[i] = keyT[a[0]][mod5(a[1] + 1)];

            str[i + 1] = keyT[a[0]][mod5(a[3] + 1)];

        }

        else if (a[1] == a[3]) {

            str[i] = keyT[mod5(a[0] + 1)][a[1]];

            str[i + 1] = keyT[mod5(a[2] + 1)][a[1]];

        }

        else {

            str[i] = keyT[a[0]][a[3]];

            str[i + 1] = keyT[a[2]][a[1]];

        }

    }

}

int mod5(int a){

    return (a % 5);

}

void encryptByPlayfairCipher(char str[], char key[]){

    char ps, ks, keyT[N][N];

    ks = strlen(key);

    ks = removeSpaces(key, ks);

    toLowerCase(key, ks);

    ps = strlen(str);

    toLowerCase(str, ps);

    ps = removeSpaces(str, ps);

    ps = prepare(str, ps);

    generateKeyTable(key, ks, keyT);

    encrypt(str, keyT, ps);

}

void decrypt(char str[], char keyT[N][N], int ps)

{

    int i, a[4];

    for (i = 0; i < ps; i += 2) {

        search(keyT, str[i], str[i + 1], a);

        if (a[0] == a[2]) {

            str[i] = keyT[a[0]][mod5(a[1] - 1)];

            str[i + 1] = keyT[a[0]][mod5(a[3] - 1)];

        }

        else if (a[1] == a[3]) {

            str[i] = keyT[mod5(a[0] - 1)][a[1]];

            str[i + 1] = keyT[mod5(a[2] - 1)][a[1]];

        }

        else {

            str[i] = keyT[a[0]][a[3]];

            str[i + 1] = keyT[a[2]][a[1]];

        }

    }

}

void decryptByPlayfairCipher(char str[], char key[])

{

    char ps, ks, keyT[N][N];

    // Key

    ks = strlen(key);

    ks = removeSpaces(key, ks);

    toLowerCase(key, ks);

    // ciphertext

    ps = strlen(str);

    toLowerCase(str, ps);

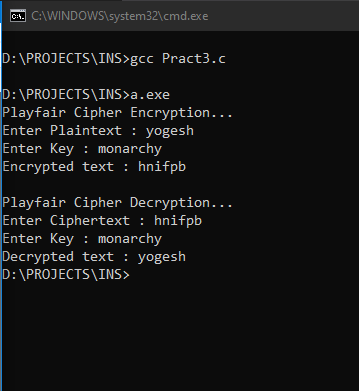
    ps = removeSpaces(str, ps);

    generateKeyTable(key, ks, keyT);

    decrypt(str, keyT, ps);

}

Output :



Experiment : 4

Aim : Implement Polyalphabetic cipher encryption-decryption.

Program (Pract4.cpp) :

#include <bits/stdc++.h>

using namespace std;

string generateKey(string str, string key);

string cipherText(string str, string key);

string originalText(string cipher\_text, string key);

int main()

{

    string str, keyword;

    cout << endl

         << "Implement Polyalphabetic cipher encryption-decryption.";

    cout << endl

         << endl

         << "Enter Keyword : ";

    cin >> keyword;

    cout << endl

         << "Enter Plain Text : ";

    cin >> str;

    string key = generateKey(str, keyword);

    string cipher\_text = cipherText(str, key);

    cout << endl

         << "Ciphertext : " << cipher\_text << endl

         << endl;

    cout << "Original/Decrypted Text : " << originalText(cipher\_text, key);

    cout << endl;

    return 0;

}

string generateKey(string str, string key)

{

    int x = str.size();

    for (int i = 0;; i++)

    {

        if (x == i)

            i = 0;

        if (key.size() == str.size())

            break;

        key.push\_back(key[i]);

    }

    return key;

}

string cipherText(string str, string key)

{

    string cipher\_text;

    for (int i = 0; i < str.size(); i++)

    {

        char x = (str[i] + key[i]) % 26;

        x += 'A';

        cipher\_text.push\_back(x);

    }

    return cipher\_text;

}

string originalText(string cipher\_text, string key)

{

    string orig\_text;

    for (int i = 0; i < cipher\_text.size(); i++)

    {

        char x = (cipher\_text[i] - key[i] + 26) % 26;

        x += 'A';

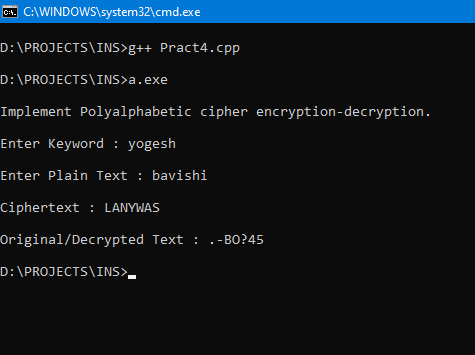
        orig\_text.push\_back(x);

    }

    return orig\_text;

}

Output :



Experiment : 5

Aim : Implement Hill cipher encryption-decryption.

Program (Pract5.cpp) :

#include <iostream>

using namespace std;

void getKeyMatrix(string, int [][3]);

void encrypt(int [][1], int [][3], int [][1]);

void HillCipher(string , string );

int main()

{

    string message, key;

    cout << endl

         << "Hill cipher Encryption-Decryption.";

    cout << endl

         << endl

         << "Enter Key : ";

    cin >> key;

    cout << endl

         << "Enter Plain Text : ";

    cin >> message;

    HillCipher(message, key);

    return 0;

}

void getKeyMatrix(string key, int keyMatrix[][3])

{

    int k = 0;

    for (int i = 0; i < 3; i++)

    {

        for (int j = 0; j < 3; j++)

        {

            keyMatrix[i][j] = key[k] - '0';

            k++;

        }

    }

}

void encrypt(int cipherMatrix[][1], int keyMatrix[][3], int messageVector[][1])

{

    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;

        }

    }

}

void HillCipher(string message, string key)

{

    int keyMatrix[3][3], cipherMatrix[3][1];

    getKeyMatrix(key, keyMatrix);

    int messageVector[3][1];

    for (int i = 0; i < 3; i++)

        messageVector[i][0] = (message[i]) % 65;

    encrypt(cipherMatrix, keyMatrix, messageVector);

    string CipherText;

    for (int i = 0; i < 3; i++)

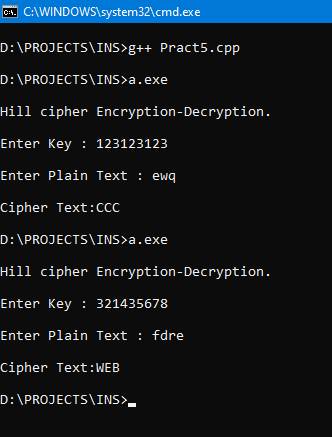
        CipherText += cipherMatrix[i][0] + 65;

    cout << endl

         << "Cipher Text:" << CipherText << endl;

}

Output:



Experiment : 6

Aim : 6 Implement Rail Fence Cipher.

Program (Pract6.cpp) :

#include <bits/stdc++.h>

using namespace std;

string encryptRailFence(string , int );

string decryptRailFence(string , int );

int main()

{

    string str;

    int rail = 0;

    cout << endl

         << "...Rail Fence Cipher Encryption ... ";

    cout << endl

         << endl

         << "Enter Plain Text : ";

    getline(cin, str);

    cout << endl

         << "Enter Rail Value : ";

    cin >> rail;

    cout << endl

         << "Cipher Text : " << encryptRailFence(str, rail) << endl;

    cout << endl

         << "...Rail Fence Cipher Decryption ... ";

    cout << endl

         << endl

         << "Enter Cipher Text : ";

    // getline(cin, str);

    cin>>str;

    cout << endl

         << "Enter Rail Value : ";

    cin >> rail;

    cout<<endl<<"Decrypted text : "<<decryptRailFence(str,rail)<<endl;

    return 0;

}

string encryptRailFence(string text, int key)

{

    char rail[key][(text.length())];

    for (int i = 0; i < key; i++)

        for (int j = 0; j < text.length(); j++)

            rail[i][j] = '\n';

    bool dir\_down = false;

    int row = 0, col = 0;

    for (int i = 0; i < text.length(); i++)

    {

        if (row == 0 || row == key - 1)

            dir\_down = !dir\_down;

        rail[row][col++] = text[i];

        dir\_down ? row++ : row--;

    }

    string result;

    for (int i = 0; i < key; i++)

        for (int j = 0; j < text.length(); j++)

            if (rail[i][j] != '\n')

                result.push\_back(rail[i][j]);

    return result;

}

string decryptRailFence(string cipher, int key)

{

    char rail[key][cipher.length()];

    for (int i = 0; i < key; i++)

        for (int j = 0; j < cipher.length(); j++)

            rail[i][j] = '\n';

    bool dir\_down;

    int row = 0, col = 0;

    for (int i = 0; i < cipher.length(); i++)

    {

        if (row == 0)

            dir\_down = true;

        if (row == key - 1)

            dir\_down = false;

        rail[row][col++] = '\*';

        dir\_down ? row++ : row--;

    }

    int index = 0;

    for (int i = 0; i < key; i++)

        for (int j = 0; j < cipher.length(); j++)

            if (rail[i][j] == '\*' && index < cipher.length())

                rail[i][j] = cipher[index++];

    string result;

    row = 0, col = 0;

    for (int i = 0; i < cipher.length(); i++)

    {

        if (row == 0)

            dir\_down = true;

        if (row == key - 1)

            dir\_down = false;

        if (rail[row][col] != '\*')

            result.push\_back(rail[row][col++]);

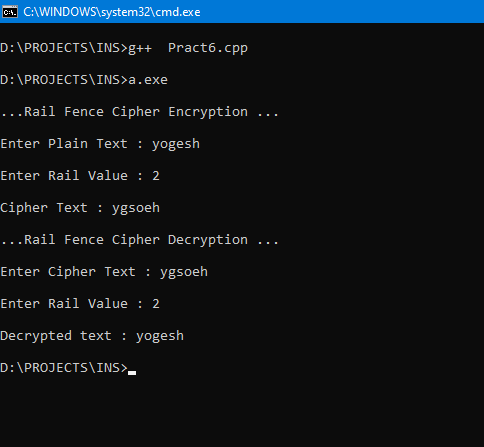
        dir\_down ? row++ : row--;

    }

    return result;

}

Output :



Experiment : 7

Aim : To implement Simple DES or AES.

Program (Pract7.cpp) :

#include <bits/stdc++.h>

using namespace std;

string hex2bin(string s);

string bin2hex(string s);

string permute(string k, int \*arr, int n);

string shift\_left(string k, int shifts);

string xor\_(string a, string b);

string encrypt(string pt, vector<string> rkb, vector<string> rk);

int main()

{

    string pt, key;

    cout << endl

         << "Simple DES ... ";

    cout << endl

         << endl

         << "Enter Key : ";

    cin >> key;

    cout << endl

         << "Enter Plain Text : ";

    cin >> pt;

    key = hex2bin(key);

    int keyp[56] = {57, 49, 41, 33, 25, 17, 9,

                    1, 58, 50, 42, 34, 26, 18,

                    10, 2, 59, 51, 43, 35, 27,

                    19, 11, 3, 60, 52, 44, 36,

                    63, 55, 47, 39, 31, 23, 15,

                    7, 62, 54, 46, 38, 30, 22,

                    14, 6, 61, 53, 45, 37, 29,

                    21, 13, 5, 28, 20, 12, 4};

    key = permute(key, keyp, 56);

    int shift\_table[16] = {1, 1, 2, 2,

                           2, 2, 2, 2,

                           1, 2, 2, 2,

                           2, 2, 2, 1};

    int key\_comp[48] = {14, 17, 11, 24, 1, 5,

                        3, 28, 15, 6, 21, 10,

                        23, 19, 12, 4, 26, 8,

                        16, 7, 27, 20, 13, 2,

                        41, 52, 31, 37, 47, 55,

                        30, 40, 51, 45, 33, 48,

                        44, 49, 39, 56, 34, 53,

                        46, 42, 50, 36, 29, 32};

    string left = key.substr(0, 28);

    string right = key.substr(28, 28);

    vector<string> rkb;

    vector<string> rk;

    for (int i = 0; i < 16; i++)

    {

        left = shift\_left(left, shift\_table[i]);

        right = shift\_left(right, shift\_table[i]);

        string combine = left + right;

        string RoundKey = permute(combine, key\_comp, 48);

        rkb.push\_back(RoundKey);

        rk.push\_back(bin2hex(RoundKey));

    }

    cout << "\nEncryption:\n\n";

    string cipher = encrypt(pt, rkb, rk);

    cout << "\nCipher Text: " << cipher << endl;

    cout << "\nDecryption\n\n";

    reverse(rkb.begin(), rkb.end());

    reverse(rk.begin(), rk.end());

    string text = encrypt(cipher, rkb, rk);

    cout << "\nPlain Text: " << text << endl;

}

string hex2bin(string s)

{

    unordered\_map<char, string> mp;

    mp['0'] = "0000";

    mp['1'] = "0001";

    mp['2'] = "0010";

    mp['3'] = "0011";

    mp['4'] = "0100";

    mp['5'] = "0101";

    mp['6'] = "0110";

    mp['7'] = "0111";

    mp['8'] = "1000";

    mp['9'] = "1001";

    mp['A'] = "1010";

    mp['B'] = "1011";

    mp['C'] = "1100";

    mp['D'] = "1101";

    mp['E'] = "1110";

    mp['F'] = "1111";

    string bin = "";

    for (int i = 0; i < s.size(); i++)

    {

        bin += mp[s[i]];

    }

    return bin;

}

string bin2hex(string s)

{

    unordered\_map<string, string> mp;

    mp["0000"] = "0";

    mp["0001"] = "1";

    mp["0010"] = "2";

    mp["0011"] = "3";

    mp["0100"] = "4";

    mp["0101"] = "5";

    mp["0110"] = "6";

    mp["0111"] = "7";

    mp["1000"] = "8";

    mp["1001"] = "9";

    mp["1010"] = "A";

    mp["1011"] = "B";

    mp["1100"] = "C";

    mp["1101"] = "D";

    mp["1110"] = "E";

    mp["1111"] = "F";

    string hex = "";

    for (int i = 0; i < s.length(); i += 4)

    {

        string ch = "";

        ch += s[i];

        ch += s[i + 1];

        ch += s[i + 2];

        ch += s[i + 3];

        hex += mp[ch];

    }

    return hex;

}

string permute(string k, int \*arr, int n)

{

    string per = "";

    for (int i = 0; i < n; i++)

    {

        per += k[arr[i] - 1];

    }

    return per;

}

string shift\_left(string k, int shifts)

{

    string s = "";

    for (int i = 0; i < shifts; i++)

    {

        for (int j = 1; j < 28; j++)

        {

            s += k[j];

        }

        s += k[0];

        k = s;

        s = "";

    }

    return k;

}

string xor\_(string a, string b)

{

    string ans = "";

    for (int i = 0; i < a.size(); i++)

    {

        if (a[i] == b[i])

        {

            ans += "0";

        }

        else

        {

            ans += "1";

        }

    }

    return ans;

}

string encrypt(string pt, vector<string> rkb, vector<string> rk)

{

    pt = hex2bin(pt);

    int initial\_perm[64] = {58, 50, 42, 34, 26, 18, 10, 2,

                            60, 52, 44, 36, 28, 20, 12, 4,

                            62, 54, 46, 38, 30, 22, 14, 6,

                            64, 56, 48, 40, 32, 24, 16, 8,

                            57, 49, 41, 33, 25, 17, 9, 1,

                            59, 51, 43, 35, 27, 19, 11, 3,

                            61, 53, 45, 37, 29, 21, 13, 5,

                            63, 55, 47, 39, 31, 23, 15, 7};

    pt = permute(pt, initial\_perm, 64);

    cout << "After initial permutation: " << bin2hex(pt) << endl;

    string left = pt.substr(0, 32);

    string right = pt.substr(32, 32);

    cout << "After splitting: L0=" << bin2hex(left)

         << " R0=" << bin2hex(right) << endl;

    int exp\_d[48] = {32, 1, 2, 3, 4, 5, 4, 5,

                     6, 7, 8, 9, 8, 9, 10, 11,

                     12, 13, 12, 13, 14, 15, 16, 17,

                     16, 17, 18, 19, 20, 21, 20, 21,

                     22, 23, 24, 25, 24, 25, 26, 27,

                     28, 29, 28, 29, 30, 31, 32, 1};

    int s[8][4][16] = {{14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    int per[32] = {16, 7, 20, 21,

                   29, 12, 28, 17,

                   1, 15, 23, 26,

                   5, 18, 31, 10,

                   2, 8, 24, 14,

                   32, 27, 3, 9,

                   19, 13, 30, 6,

                   22, 11, 4, 25};

    cout << endl;

    for (int i = 0; i < 16; i++)

    {

        string right\_expanded = permute(right, exp\_d, 48);

        string x = xor\_(rkb[i], right\_expanded);

        string op = "";

        for (int i = 0; i < 8; i++)

        {

            int row = 2 \* int(x[i \* 6] - '0') + int(x[i \* 6 + 5] - '0');

            int col = 8 \* int(x[i \* 6 + 1] - '0') + 4 \* int(x[i \* 6 + 2] - '0') + 2 \* int(x[i \* 6 + 3] - '0') + int(x[i \* 6 + 4] - '0');

            int val = s[i][row][col];

            op += char(val / 8 + '0');

            val = val % 8;

            op += char(val / 4 + '0');

            val = val % 4;

            op += char(val / 2 + '0');

            val = val % 2;

            op += char(val + '0');

        }

        op = permute(op, per, 32);

        x = xor\_(op, left);

        left = x;

        if (i != 15)

        {

            swap(left, right);

        }

        cout << "Round " << i + 1 << " " << bin2hex(left) << " "

             << bin2hex(right) << " " << rk[i] << endl;

    }

    string combine = left + right;

    int final\_perm[64] = {40, 8, 48, 16, 56, 24, 64, 32,

                          39, 7, 47, 15, 55, 23, 63, 31,

                          38, 6, 46, 14, 54, 22, 62, 30,

                          37, 5, 45, 13, 53, 21, 61, 29,

                          36, 4, 44, 12, 52, 20, 60, 28,

                          35, 3, 43, 11, 51, 19, 59, 27,

                          34, 2, 42, 10, 50, 18, 58, 26,

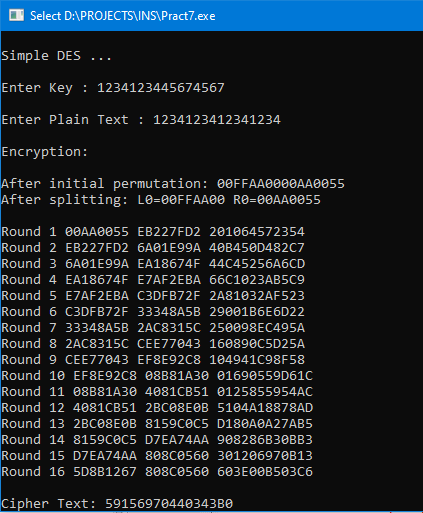
                          33, 1, 41, 9, 49, 17, 57, 25};

    string cipher = bin2hex(permute(combine, final\_perm, 64));

    return cipher;

}

Output :



Experiment : 8

Aim : Implement Diffi-Hellmen Key exchange Method.

Program (Pract8.cpp) :

#include <iostream>

using namespace std;

int compute(int, int, int);

int main()

{

    int prime, root;

    int privateKeyA, privateKeyB;

    int A, B;

    cout << endl

         << "Implement Diffi-Hellmen Key exchange Method" << endl;

    cout << endl

         << "Enter Prime Number : ";

    cin >> prime;

    cout << endl

         << "Enter Root : ";

    cin >> root;

    cout << endl

         << "Enter Private Key of User A : ";

    cin >> privateKeyA;

    cout << endl

         << "Enter Private Key of User B : ";

    cin >> privateKeyB;

    A = compute(root, privateKeyA, prime);

    B = compute(root, privateKeyB, prime);

    int keyA = compute(B, privateKeyA, prime);

    int keyB = compute(A, privateKeyB, prime);

    cout << endl;

    cout << "A's Secret Key is " << keyA << endl

         << "B's Secret Key is " << keyB;

    cout << endl;

    return 0;

}

int compute(int a, int m, int n)

{

    int r;

    int y = 1;

    while (m > 0)

    {

        r = m % 2;

        if (r == 1)

            y = (y \* a) % n;

        a = a \* a % n;

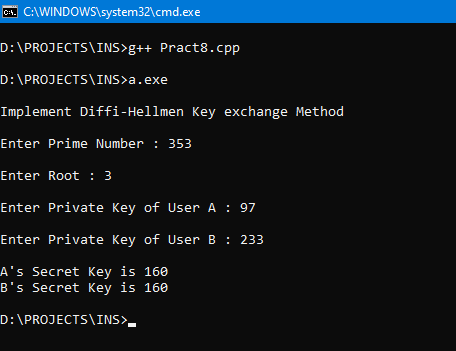
        m = m / 2;

    }

    return y;

}

Output :



Experiment : 9

Aim : Implement RSA encryption-decryption algorithm.

Program (Pract9.cpp) :

#include <stdio.h>

#include <iostream>

#include <math.h>

using namespace std;

void encrypt();

void decrypt();

long int multipicative\_inverse(long int, long int);

void key\_pair(long int , long int , long int );

int main()

{

    int choice;

    cout << endl

         << "RSA encryption-decryption algorithm ... ";

    while (1)

    {

        cout << endl

             << endl;

        cout << "1.Encrypt\t2.Decrypt\t3.Exit" << endl;

        cout << endl

             << "Enter your choice: ";

        cin >> choice;

        switch (choice)

        {

        case 1:

            encrypt();

            break;

        case 2:

            decrypt();

            break;

        case 3:

            exit(0);

        default:

            cout << endl;

            cout << "Invalid choice,Enter valid choice : " << endl;

        }

    }

    return 0;

}

void key\_pair(long int e, long int n, long int d)

{

    cout << endl

         << "Encryption key(e,O(n)): (" << e << "," << n << ")" << endl;

    cout << endl

         << "Decryption key(d,O(n)): (" << d << "," << n << ")" << endl;

}

void encrypt()

{

    long int pt, p, q, n, fn, e, temp, d, ct;

    cout << endl

         << "Enter your message to encrypt : ";

    cin >> pt;

    cout << endl

         << "Enter two prime numbers p and q: ";

    cin >> p >> q;

    cout << endl

         << "Enter value of e: ";

    cin >> e;

    n = p \* q;

    fn = (p - 1) \* (q - 1);

    d = multipicative\_inverse(fn, e);

    if (d == 0)

    {

        cout << endl;

        cout << "RSA is not possible for given p and q." << endl;

        exit(0);

    }

    ct = fmod(pow(pt, e), n);

    key\_pair(e, fn, d);

    cout << endl

         << "Cipher text: " << ct << endl;

}

void decrypt()

{

    long int pt, d, ct, n;

    cout << endl

         << "Enter message to decrypt: ";

    cin >> ct;

    cout << endl

         << "Enter key pair(d,n): ";

    cin >> d >> n;

    pt = fmod(pow(ct, d), n);

    cout << endl

         << "Plain Text : " << pt << endl;

}

long int multipicative\_inverse(long int m, long int b)

{

    long int a1 = 1, a2 = 0, a3 = m, b1 = 0, b2 = 1, b3 = b, t1, t2, t3, q;

top:

    if (b3 == 0)

        return 0;

    else if (b3 == 1)

    {

        if (b2 < 0)

            return (b2 + m);

        else

            return b2;

    }

    q = a3 / b3;

    t1 = b1;

    t2 = b2;

    t3 = b3;

    b1 = a1 - (q \* b1);

    b2 = a2 - (q \* b2);

    b3 = a3 - (q \* b3);

    a1 = t1;

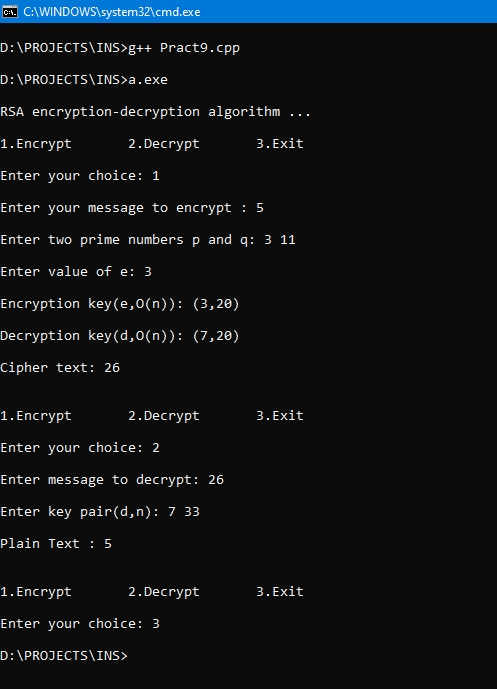
    a2 = t2;

    a3 = t3;

    goto top;

}

Output :



Experiment : 10

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

Program (Pract10.java) :

import java.math.BigInteger;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.util.Scanner;

public class Pract10 {

  public static String encryptThisString(String input) {

    try {

      MessageDigest md = MessageDigest.getInstance("SHA-1");

      byte[] messageDigest = md.digest(input.getBytes());

      BigInteger no = new BigInteger(1, messageDigest);

      String hashtext = no.toString(16);

      while (hashtext.length() < 32) {

        hashtext = "0" + hashtext;

      }

      return hashtext;

    } catch (NoSuchAlgorithmException e) {

      throw new RuntimeException(e);

    }

  }

  public static void main(String args[]) throws NoSuchAlgorithmException {

    String str;

    Scanner input = new Scanner(System.in);

    System.out.println("\nSHA-1 Algorithm ");

    System.out.print("\nEnter Plain Text : ");

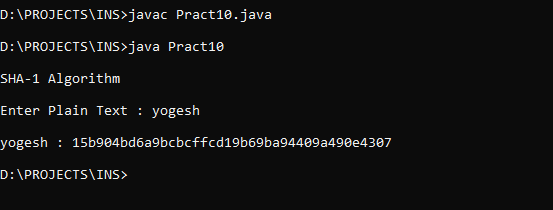
    str = input.nextLine();

    System.out.println("\n" + str + " : " + encryptThisString(str));

  }

}

Output :



Experiment : 11

Aim : Implement a digital signature algorithm.

Program (Pract11.java) :

import java.security.KeyPair;

import java.security.KeyPairGenerator;

import java.security.PrivateKey;

import java.security.Signature;

import java.util.Scanner;

public class Pract11 {

    public static void main(String args[]) throws Exception {

        Scanner sc = new Scanner(System.in);

        System.out.println("\nDigital Signature Algorithm(DSA) ");

        System.out.print("\nEnter Plain Text : ");

        String msg = sc.nextLine();

        KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("DSA");

        keyPairGen.initialize(2048);

        KeyPair pair = keyPairGen.generateKeyPair();

        PrivateKey privKey = pair.getPrivate();

        Signature sign = Signature.getInstance("SHA256withDSA");

        sign.initSign(privKey);

        byte[] bytes = "msg".getBytes();

        sign.update(bytes);

        byte[] signature = sign.sign();

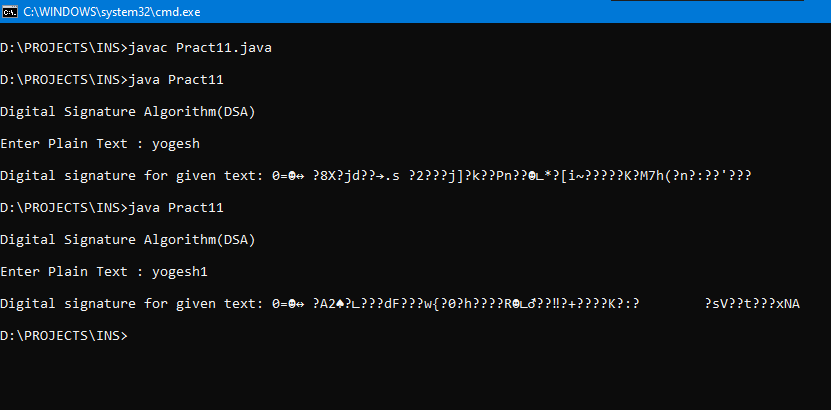
        String dot = new String(signature,"UTF-8");

        System.out.println("\nDigital signature for given text: "+dot);

    }

}

Output :



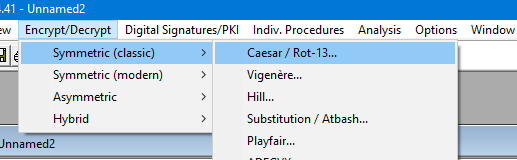
Experiment : 12

Aim : Perform various encryption-decryption techniques with cryptool.

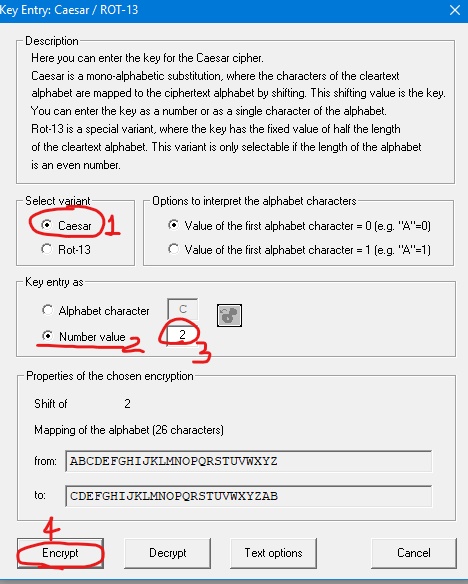
Steps :

1. Encryption and Decryption of Caesar Cipher:

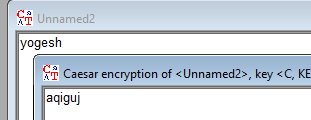
* Create new file by clicking **File->New** option.
* New empty file will be created. Enter your PlainText there.
* Click **Encrypt/Decrypt->Symmetric(classic)->Caeser/Rot-13…**  as shown in image below.



* In encryption, we are replacing the plaintext letter with the 2rd letter of the alphabet that is if “A” is our plaintext character, then the Ciphertext will be “C”.

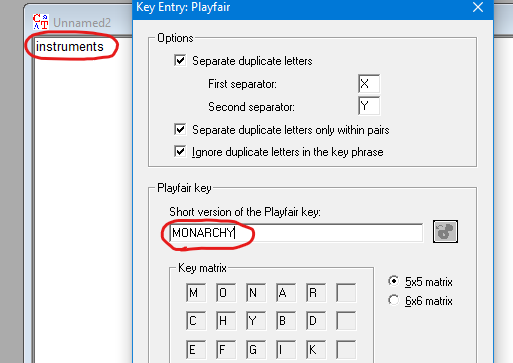


* Output shows up in new window like below image.

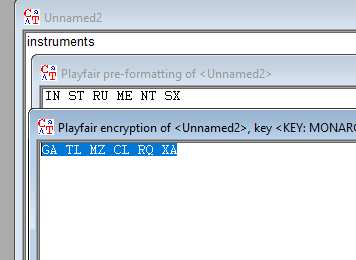


1. Encryption and Decryption of Playfair Cipher:

* Again, we have to move to **Encrypt/Decrypt -> Symmetric -> Playfair…** and perform the encryption part. We are putting the plaintext – instruments.

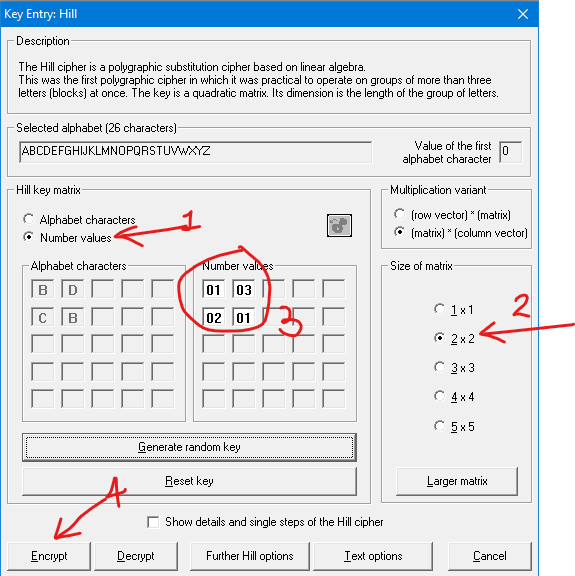


* Output shown in new window as “GA TL MZ CL RQ XA”.

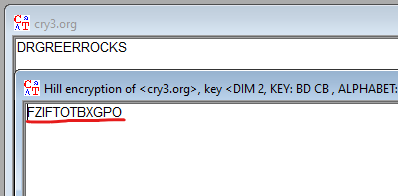


1. Encryption and Decryption of Hill Cipher:

* Again, we have to move to **Encrypt/Decrypt -> Symmetric -> Hill…** and perform the encryption part. We are putting the plaintext as – DRGREERROCKS and assuming that the program gives us the Ciphertext as – FZIFTOTBXGPO.

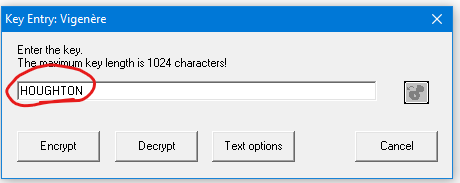


* Output shows in new window as shown in image below.

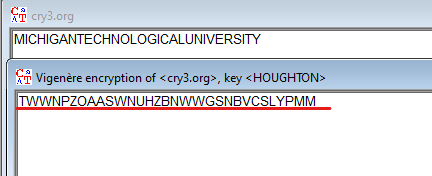


1. Encryption and Decryption of Vigener Cipher:

* Again, we have to move to **Encrypt/Decrypt -> Symmetric -> Vigener…** and perform the encryption part. We are putting the plaintext as – MICHIGANTECHNOLOGICALUNIVERSITY and program gives us the Ciphertext as – TWWNPZOAAS…..,with the help of key as – HOUGHTON.



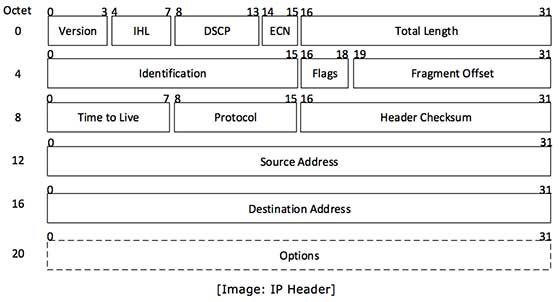
* Output shows in new windows as shown below.



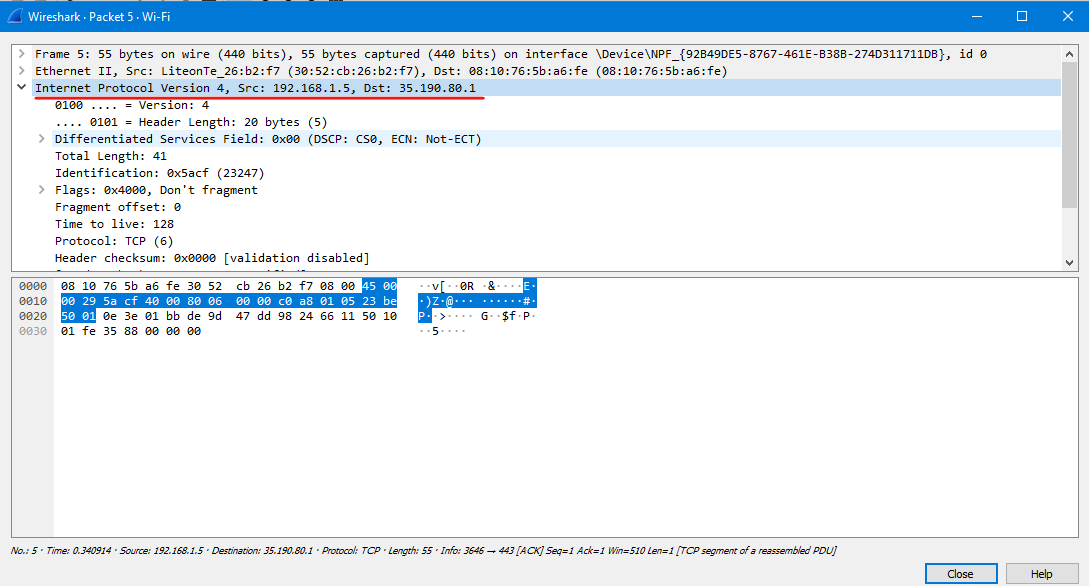
Experiment : 13

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

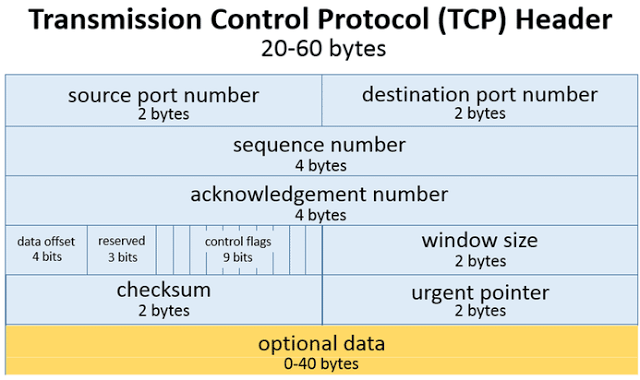
* Description:
* Wireshark is a free and open source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education. Originally named Ethereal, the project was renamed Wireshark in May 2006 due to trademark issues.
* Wireshark is cross-platform, using the Qt widget toolkit in current releases to implement its user interface, and using pcap to capture packets; it runs on Linux, macOS, BSD, Solaris, some other Unix-like operating systems, and Microsoft Windows.
* There is also a terminal-based (non-GUI) version called TShark. Wireshark, and the other programs distributed with it such as TShark, are free software, released under the terms of the GNU General Public License.
* Studing TCP/UDP Using WireShark
* **Internet Protocol**
* The Internet Protocol (IP) is the method or protocol by which data is sent from one computer to another on the Internet. Each computer (known as a host) on the Internet has at least one IP address that uniquely identifies from all other computers on the Internet.



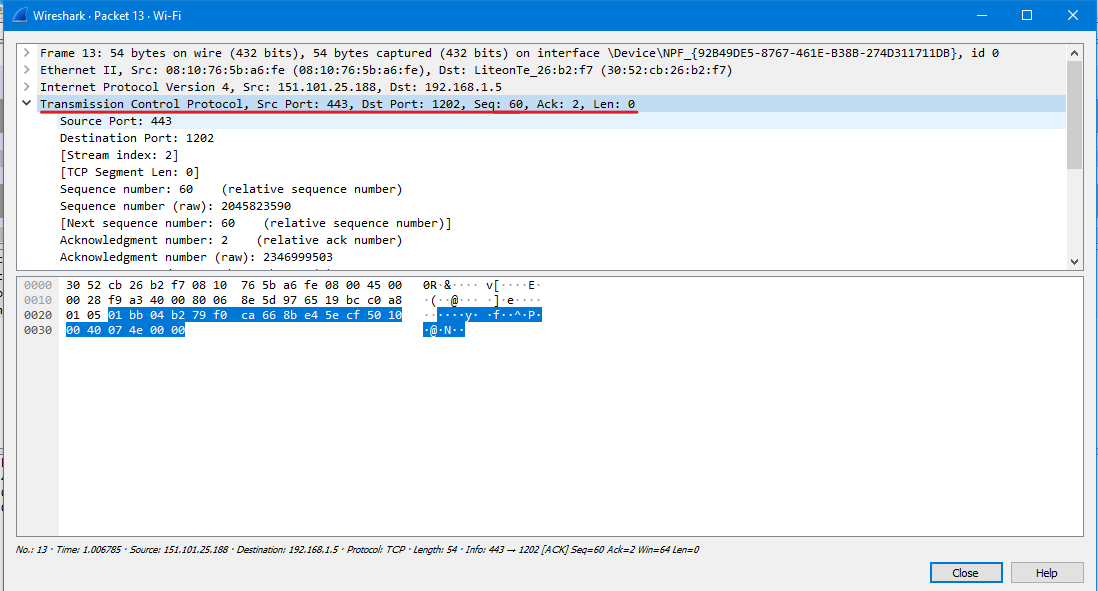
* **Packet Analysing**

****

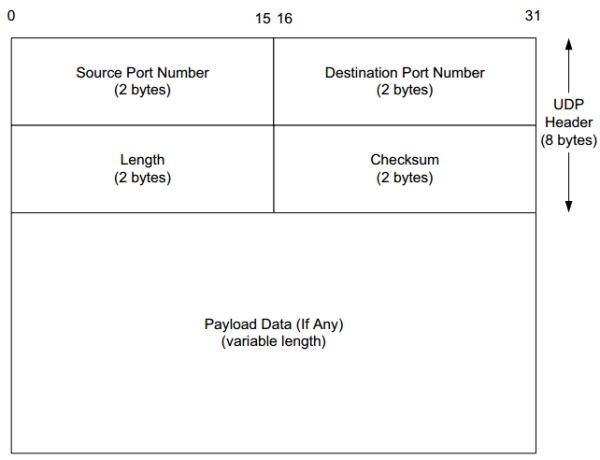
* **TCP Frame format:**
* The Transmission Control Protocol (TCP) is a core protocol of the Internet protocol suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP). Therefore, the entire suite is commonly referred to as TCP/IP.

****

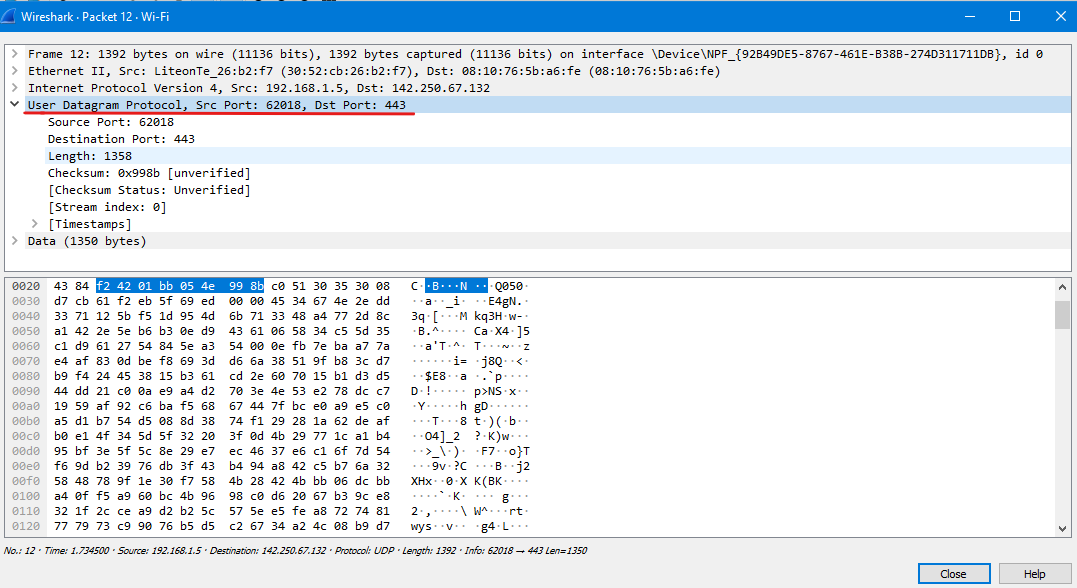
* **Packet Analyzing**

****

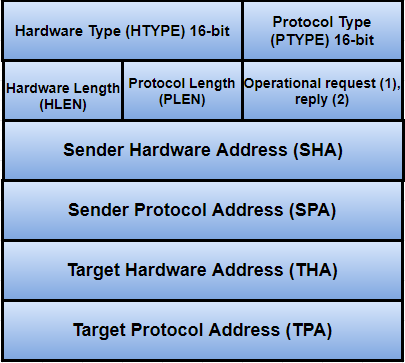
* **UDP frame format:**
* UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes any unreliability of the underlying network protocol to the user's program.



* **Packet Analyzing**

****

* **Address Resolution Protocol (ARP)**
* Address Resolution Protocol (ARP) is one of the major protocol in the TCP/IP suit and the purpose of Address Resolution Protocol (ARP) is to resolve an IPv4 address (32 bit Logical Address) to the physical address (48 bit MAC Address). Network Applications at the Application Layer use IPv4 Address to communicate with another device.



* **Packet Analyzing**

