**Cryptography and Network Security Lab (6CS451)**

**Assignment No 1**

**Name : Vijay Sambhaji Mali**

**PRN : 22520008**

1. **Perform encryption, decryption using the following substitution techniques a. Ceaser cipher, b. playfair cipher c. Hill Cipher d. Vigenere cipher.**
2. **Ceaser Cipher :**

CODE :

#include <iostream>

using namespace std;

string encrypt(string text, int s)

{

    string result = "";

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

        if (isupper(text[i]))

            result += char(int(text[i] + s - 65) % 26 + 65);

        else

            result += char(int(text[i] + s - 97) % 26 + 97);

    }

    return result;

}

int main()

{

    string text = "ATTACKATONCE";

    int s = 4;

    cout << "Text : " << text;

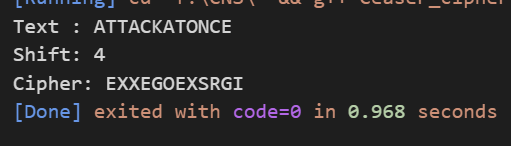
    cout << "\nShift: " << s;

    cout << "\nCipher: " << encrypt(text, s);

    return 0;

}

OUTPUT :



1. **Playfair Cipher :**

CODE :

#include <bits/stdc++.h>

using namespace std;

#define SIZE 30

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[5][5]) {

    int i, j, k, flag = 0;

    int dicty[26] = { 0 };

    for (i = 0; i < ks; i++) {

        if (key[i] != '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[5][5], char a, char b, int arr[]) {

    int i, j;

    if (a == 'j')

        a = 'i';

    else if (b == 'j')

        b = 'i';

    for (i = 0; i < 5; i++) {

        for (j = 0; j < 5; 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 mod5(int a) { return (a % 5); }

int prepare(char str[], int ptrs) {

    if (ptrs % 2 != 0) {

        str[ptrs++] = 'z';

        str[ptrs] = '\0';

    }

    return ptrs;

}

void encrypt(char str[], char keyT[5][5], 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 encryptByPlayfairCipher(char str[], char key[]) {

    char ps, ks, keyT[5][5];

    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);

}

int main() {

    char str[SIZE], key[SIZE];

    strcpy(key, "Monarchy");

    cout << "Key text: " << key << "\n";

    strcpy(str, "instruments");

    cout << "Plain text: " << str << "\n";

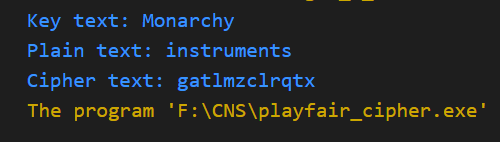
    encryptByPlayfairCipher(str, key);

    cout << "Cipher text: " << str << "\n";

    return 0;

}

OUTPUT :



1. Hill Cipher

**Input :**

**Plaintext: ACT**

**Key: GYBNQKURP**

CODE :

// C++ code to implement Hill Cipher

#include <iostream>

using namespace std;

// Following function generates the

//  key matrix for the key string

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]) % 65;

            k++;

        }

    }

}

// Following function encrypts the message

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;

        }

    }

}

// Function to implement Hill Cipher

void HillCipher(string message, string key)

{

    int keyMatrix[3][3];

    getKeyMatrix(key, keyMatrix);

    int messageVector[3][1];

    for (int i = 0; i < 3; i++)

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

    int cipherMatrix[3][1];

    encrypt(cipherMatrix, keyMatrix, messageVector);

    string CipherText;

    for (int i = 0; i < 3; i++)

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

    cout << " Ciphertext:" << CipherText;

}

int main()

{

    string message = "ACT";

    string key = "GYBNQKURP";

    HillCipher(message, key);

    return 0;

}

OUTPUT :



d. Vigenere cipher :

CODE :

// C++ code to implement Vigenere Cipher

#include <bits/stdc++.h>

using namespace std;

// This function generates the key in

// a cyclic manner until it's length isn't

// equal to the length of original text

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;

}

// This function returns the encrypted text

// generated with the help of the key

string cipherText(string str, string key)

{

    string cipher\_text;

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

        // converting in range 0-25

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

        // convert into alphabets(ASCII)

        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++) {

        // converting in range 0-25

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

        x += 'A';

        orig\_text.push\_back(x);

    }

    return orig\_text;

}

int main()

{

    string str = "hELLO";

    string keyword = "AYUSH";

    if (any\_of(str.begin(), str.end(), ::islower))

        transform(str.begin(), str.end(), str.begin(),

                  ::toupper);

    if (any\_of(keyword.begin(), keyword.end(), ::islower))

        transform(keyword.begin(), keyword.end(),

                  keyword.begin(), ::toupper);

    string key = generateKey(str, keyword);

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

    cout << "Ciphertext : " << cipher\_text << "\n";

    cout << "Original/Decrypted Text : "

         << originalText(cipher\_text, key);

    return 0;

}

OUTPUT :

