**Cryptography, Network and Security**

Assignment 1

1. Perform encryption, decryption using the following substitution techniques

a. Ceaser cipher,

b. playfair cipher

c. Hill Cipher

d. Vigenere cipher

Code:

a.

#include <iostream>

using namespace std;

*// Function to encrypt text using Caesar Cipher*

string caesarEncrypt(string text, int s)

{

    string result = "";

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

    {

        if (text[i] == ' ')

        {

            result += ' ';

            continue;

        }

        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;

}

*// Function to decrypt text using Caesar Cipher*

string caesarDecrypt(string text, int s)

{

    return caesarEncrypt(text, 26 - s);

}

int main()

{

    cout << "Enter the text to be encrypted: ";

    string text;

    getline(cin, text);

    cout << "Enter the shift value: ";

    int s;

    cin >> s;

    cout << "Original Text: " << text << endl;

    string encrypted = caesarEncrypt(text, s);

    cout << "Encrypted Text: " << encrypted << endl;

    cout << "Decrypted Text: " << caesarDecrypt(encrypted, s) << endl;

    return 0;

}

b.

#include <iostream>

#include <string>

#include <algorithm>

using namespace std;

char keyMatrix[5][5];

*// Function to remove duplicates from a string*

string removeDuplicates(string str) {

    string result;

    bool used[26] = {false};

    for (char c : str) {

        if (!used[c - 'a']) {

            used[c - 'a'] = true;

            result += c;

        }

    }

    return result;

}

*// Function to create the Playfair key matrix*

void generateKeyMatrix(string key) {

    key = removeDuplicates(key);

    key.erase(remove(key.begin(), key.end(), 'j'), key.end());

    bool used[26] = {false};

    int k = 0;

    for (char c : key) {

        used[c - 'a'] = true;

    }

    key += "abcdefghiklmnopqrstuvwxyz";

    int index = 0;

    for (int i = 0; i < 5; i++) {

        for (int j = 0; j < 5; j++) {

            while (used[key[index] - 'a']) index++;

            keyMatrix[i][j] = key[index];

            used[key[index] - 'a'] = true;

        }

    }

}

*// Function to find the position of a character in the key matrix*

void findPosition(char c, int *&*row, int *&*col) {

    for (int i = 0; i < 5; i++) {

        for (int j = 0; j < 5; j++) {

            if (keyMatrix[i][j] == c) {

                row = i;

                col = j;

                return;

            }

        }

    }

}

*// Function to preprocess the plaintext (handle repeated characters, make pairs)*

string preprocessPlaintext(string text) {

    text.erase(remove(text.begin(), text.end(), ' '), text.end());

    for (int i = 0; i < text.length(); i += 2) {

        if (i + 1 == text.length()) {

            text += 'x';

        } else if (text[i] == text[i + 1]) {

            text.insert(i + 1, 1, 'x');

        }

    }

    return text;

}

*// Function to encrypt plaintext using Playfair Cipher*

string playfairEncrypt(string plaintext, string key) {

    generateKeyMatrix(key);

    plaintext = preprocessPlaintext(plaintext);

    string encrypted = "";

    for (int i = 0; i < plaintext.length(); i += 2) {

        int r1, c1, r2, c2;

        findPosition(plaintext[i], r1, c1);

        findPosition(plaintext[i + 1], r2, c2);

        if (r1 == r2) {

            encrypted += keyMatrix[r1][(c1 + 1) % 5];

            encrypted += keyMatrix[r2][(c2 + 1) % 5];

        } else if (c1 == c2)

            encrypted += keyMatrix[(r1 + 1) % 5][c1];

            encrypted += keyMatrix[(r2 + 1) % 5][c2];

        } else {

            encrypted += keyMatrix[r1][c2];

            encrypted += keyMatrix[r2][c1];

        }

    }

    return encrypted;

}

*// Function to decrypt ciphertext using Playfair Cipher*

string playfairDecrypt(string ciphertext, string key) {

    generateKeyMatrix(key);

    string decrypted = "";

    for (int i = 0; i < ciphertext.length(); i += 2) {

        int r1, c1, r2, c2;

        findPosition(ciphertext[i], r1, c1);

        findPosition(ciphertext[i + 1], r2, c2);

        if (r1 == r2) {

            decrypted += keyMatrix[r1][(c1 + 4) % 5];

            decrypted += keyMatrix[r2][(c2 + 4) % 5];

        } else if (c1 == c2) {

            decrypted += keyMatrix[(r1 + 4) % 5][c1];

            decrypted += keyMatrix[(r2 + 4) % 5][c2];

        } else

            decrypted += keyMatrix[r1][c2];

            decrypted += keyMatrix[r2][c1];

        }

    }

    return decrypted;

}

int main() {

    cout<<"Enter the single word key: ";

    string key;

    cin>>key;

    cout<<"Enter the plaintext: ";

    string plaintext;

    getline(cin>>ws, plaintext);

    cout << "Original Text: " << plaintext << endl;

    string encrypted = playfairEncrypt(plaintext, key);

    cout << "Encrypted Text: " << encrypted << endl;

    string decrypted = playfairDecrypt(encrypted, key);

    cout << "Decrypted Text: " << decrypted << endl;

    return 0;

}

c.

#include <iostream>

#include <vector>

using namespace std;

*// Function to multiply matrices*

vector<int> matrixMultiplication(vector<vector<int>> key, vector<int> textVec, int n)

{

    vector<int> result(n);

    for (int i = 0; i < n; i++)

    {

        result[i] = 0;

        for (int j = 0; j < n; j++)

        {

            result[i] += key[i][j] \* textVec[j];

        }

        result[i] = result[i] % 26;

    }

    return result;

}

*// Function to encrypt using Hill cipher*

string hillEncrypt(string message, vector<vector<int>> key)

{

    int n = key.size();

    vector<int> textVec(n);

    for (int i = 0; i < n; i++)

    {

        textVec[i] = message[i] - 'A';

    }

    vector<int> cipherVec = matrixMultiplication(key, textVec, n);

    string cipherText = "";

    for (int i = 0; i < n; i++)

    {

        cipherText += cipherVec[i] + 'A';

    }

    return cipherText;

}

int main()

{

    string message = "ACT";

    vector<vector<int>> key = {{6, 24, 1}, {13, 16, 10}, {20, 17, 15}};

    cout << "Original Text: " << message << endl;

    string encrypted = hillEncrypt(message, key);

    cout << "Encrypted Text: " << encrypted << endl;

    return 0;

}

d.

#include <iostream>

using namespace std;

*// Function to generate key to match length of text*

string generateKey(string text, string key)

{

    int x = text.size();

    for (int i = 0;; i++)

    {

        if (x == i)

            i = 0;

        if (key.size() == text.size())

            break;

        key.push\_back(key[i]);

    }

    return key;

}

*// Function to encrypt using Vigenere Cipher*

string vigenereEncrypt(string text, string key)

{

    string encryptedText;

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

    {

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

        x += 'A';

        encryptedText.push\_back(x);

    }

    return encryptedText;

}

*// Function to decrypt using Vigenere Cipher*

string vigenereDecrypt(string encryptedText, string key)

{

    string decryptedText;

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

    {

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

        x += 'A';

        decryptedText.push\_back(x);

    }

    return decryptedText;

}

int main()

{

    cout<<"Enter the text to be encrypted: ";

    string text;

    getline(cin>>ws, text);

    cout<<"Enter the key: ";

    string key;

    cin>>key;

    key = generateKey(text, key);

    string encrypted = vigenereEncrypt(text, key);

    cout << "Original Text: " << text << endl;

    cout << "Encrypted Text: " << encrypted << endl;

    cout << "Decrypted Text: " << vigenereDecrypt(encrypted, key) << endl;

    return 0;

}