**Cryptography and Network Security**

**Lab**

**Assignment No. 1**

**Name : Sourabh Shankar Patil**

**PRN : 21510045**

**Batch : B2**

**1. Caesar Cipher**

The **Caesar Cipher** is a simple substitution cipher where each letter in the plaintext is shifted by a certain number of positions down the alphabet. Named after Julius Caesar, who used it to communicate with his officials, this cipher replaces each letter with the letter that is a fixed number of positions away. For example, with a shift of 3, 'A' would become 'D', 'B' becomes 'E', and so on. It’s a monoalphabetic cipher, meaning each letter corresponds to only one letter in the ciphertext.

**Example:**

* Plaintext: "HELLO"
* Shift: 3
* Ciphertext: "KHOOR"

**Weakness:** The Caesar Cipher is vulnerable to frequency analysis and brute-force attacks since it has only 25 possible shifts.

#include <bits/stdc++.h>

#include <string>

using namespace std;

// Function to encrypt the text using Caesar Cipher

string caesarCipherEncrypt(string text, int shift)

{

    string result = "";

    // Traverse the text

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

    {

        char ch = text[i];

        // Encrypt uppercase letters

        if (isupper(ch))

        {

            result += char(int(ch + shift - 65) % 26 + 65);

        }

        // Encrypt lowercase letters

        else if (islower(ch))

        {

            result += char(int(ch + shift - 97) % 26 + 97);

        }

        // If it's not an alphabet, keep it unchanged

        else

        {

            result += ch;

        }

    }

    return result;

}

// Function to decrypt the text using Caesar Cipher

string caesarCipherDecrypt(string text, int shift)

{

    string result = "";

    // Traverse the text

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

    {

        char ch = text[i];

        // Decrypt uppercase letters

        if (isupper(ch))

        {

            result += char(int(ch - shift - 65 + 26) % 26 + 65);

        }

        // Decrypt lowercase letters

        else if (islower(ch))

        {

            result += char(int(ch - shift - 97 + 26) % 26 + 97);

        }

        // If it's not an alphabet, keep it unchanged

        else

        {

            result += ch;

        }

    }

    return result;

}

int main()

{

    string text;

    int shift;

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

    getline(cin, text);

    cout << "Enter the shift value: ";

    cin >> shift;

    string encryptedText = caesarCipherEncrypt(text, shift);

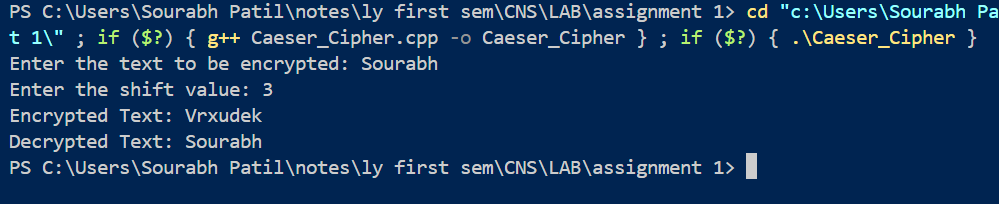
    string decryptedText = caesarCipherDecrypt(encryptedText, shift);

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

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

    return 0;

}

****

**2. Hill Cipher**

The **Hill Cipher** is a polygraphic substitution cipher that uses linear algebra to encrypt a block of letters (usually two or three) at once. The cipher uses a matrix (key) to transform each block of plaintext into ciphertext. The plaintext letters are represented as vectors, and the encryption process involves multiplying these vectors by the key matrix.

To decrypt the message, the inverse of the key matrix is used. The strength of the Hill Cipher lies in the use of matrix mathematics, which adds complexity to the encryption.

**Example:**

* Plaintext: "ACT" (represented as vectors)
* Key matrix: A 2x2 matrix like [[3, 3], [2, 5]]
* Ciphertext: The result of matrix multiplication

**Weakness:** It requires an invertible matrix key, and the method can be broken if enough plaintext-ciphertext pairs are known.

#include <iostream>

#include <vector>

using namespace std;

vector<vector<int>> keyMatrix = {{3, 3}, {2, 5}};

vector<int> multiplyMatrix(vector<int> vec)

{

    vector<int> result(2);

    result[0] = (keyMatrix[0][0] \* vec[0] + keyMatrix[0][1] \* vec[1]) % 26;

    result[1] = (keyMatrix[1][0] \* vec[0] + keyMatrix[1][1] \* vec[1]) % 26;

    return result;

}

string hillEncrypt(string text)

{

    string result = "";

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

    {

        vector<int> vec = {text[i] - 'A', text[i + 1] - 'A'};

        vector<int> resVec = multiplyMatrix(vec);

        result += (char)(resVec[0] + 'A');

        result += (char)(resVec[1] + 'A');

    }

    return result;

}

int main()

{

    string text = "HELP";

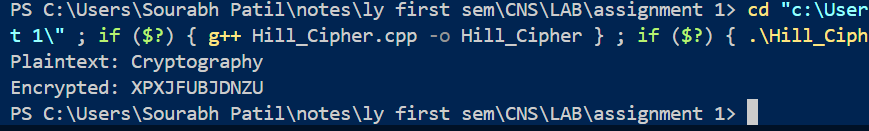
    string encrypted = hillEncrypt(text);

    cout << "Plaintext: " << text << endl;

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

    return 0;

}

****

**3. Playfair Cipher**

The **Playfair Cipher** is a digraph substitution cipher, meaning it encrypts two letters at a time. The cipher uses a 5x5 grid of letters (with 'I' and 'J' typically sharing a space) to replace each pair of letters in the plaintext with another pair based on their positions in the grid.

The rules for encryption depend on whether the letters appear in the same row, column, or different rows/columns. If in the same row, each letter is replaced with the one to its right; if in the same column, with the one below it; and if neither, the two letters form a rectangle and are replaced by the letters on the same row in the corners of the rectangle.

**Example:**

* Key: "MONARCHY" (to form the grid)
* Plaintext: "HELLO"
* Ciphertext: "GCNVF"

**Weakness:** It is vulnerable to frequency analysis of digraphs and can be broken with known-plaintext attacks.

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

using namespace std;

// Function to remove duplicates from a string

string removeDuplicates(string str)

{

    string result;

    for (char c : str)

    {

        if (result.find(c) == string::npos)

        {

            result += c;

        }

    }

    return result;

}

// Function to prepare the 5x5 key matrix

void generateKeyMatrix(string key, char keyMatrix[5][5])

{

    key = removeDuplicates(key);                               // Remove duplicates from the key

    key.erase(remove(key.begin(), key.end(), 'J'), key.end()); // Remove 'J'

    vector<bool> used(26, false);

    used['J' - 'A'] = true; // Treat 'I' and 'J' as the same letter

    int index = 0;

    for (char c : key)

    {

        if (!used[c - 'A'])

        {

            keyMatrix[index / 5][index % 5] = c;

            used[c - 'A'] = true;

            index++;

        }

    }

    // Fill the remaining spaces with other letters

    for (char c = 'A'; c <= 'Z'; c++)

    {

        if (!used[c - 'A'])

        {

            keyMatrix[index / 5][index % 5] = c;

            index++;

        }

    }

}

// Function to find the position of a letter in the key matrix

void findPosition(char keyMatrix[5][5], char c, int &row, int &col)

{

    if (c == 'J')

        c = 'I'; // Treat 'I' and 'J' as the same letter

    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 encrypt a pair of characters

string encryptPair(char keyMatrix[5][5], char a, char b)

{

    int row1, col1, row2, col2;

    findPosition(keyMatrix, a, row1, col1);

    findPosition(keyMatrix, b, row2, col2);

    if (row1 == row2)

    {

        // Same row, shift columns right

        return string(1, keyMatrix[row1][(col1 + 1) % 5]) + keyMatrix[row2][(col2 + 1) % 5];

    }

    else if (col1 == col2)

    {

        // Same column, shift rows down

        return string(1, keyMatrix[(row1 + 1) % 5][col1]) + keyMatrix[(row2 + 1) % 5][col2];

    }

    else

    {

        // Rectangle swap

        return string(1, keyMatrix[row1][col2]) + keyMatrix[row2][col1];

    }

}

// Function to prepare the plaintext by handling pairs

string prepareText(string text)

{

    string result;

    for (size\_t i = 0; i < text.length(); i++)

    {

        result += toupper(text[i]);

        if (i + 1 < text.length() && toupper(text[i]) == toupper(text[i + 1]))

        {

            result += 'X'; // Insert 'X' between duplicate letters in a pair

        }

    }

    if (result.length() % 2 != 0)

    {

        result += 'X'; // Append 'X' if the length of the text is odd

    }

    return result;

}

// Function to encrypt the plaintext using Playfair cipher

string playfairEncrypt(string plaintext, string key)

{

    char keyMatrix[5][5];

    generateKeyMatrix(key, keyMatrix);

    plaintext = prepareText(plaintext);

    string cipherText = "";

    for (size\_t i = 0; i < plaintext.length(); i += 2)

    {

        cipherText += encryptPair(keyMatrix, plaintext[i], plaintext[i + 1]);

    }

    return cipherText;

}

int main()

{

    string plaintext, key;

    cout << "Enter the plaintext: ";

    getline(cin, plaintext);

    cout << "Enter the key: ";

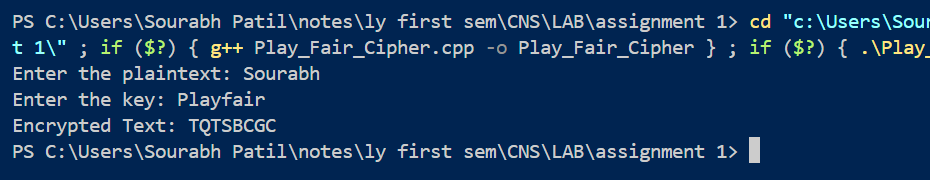
    getline(cin, key);

    string cipherText = playfairEncrypt(plaintext, key);

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

    return 0;

}

****

**4. Vigenère Cipher**

The **Vigenère Cipher** is a polyalphabetic substitution cipher that uses a keyword to determine the shift for each letter in the plaintext. Each letter in the plaintext is shifted by a different amount based on the corresponding letter in the keyword. The key is repeated as needed to match the length of the plaintext.

Unlike the Caesar Cipher, which uses a constant shift, the Vigenère Cipher uses multiple shifts, making it more resistant to simple frequency analysis. However, it can still be broken with techniques like the Kasiski examination.

**Example:**

* Plaintext: "ATTACKATDAWN"
* Key: "LEMON"
* Ciphertext: "LXFOPVEFRNHR"

**Weakness:** While stronger than the Caesar Cipher, the Vigenère Cipher is vulnerable to modern cryptanalysis techniques.

These ciphers represent important milestones in cryptography, ranging from simple to more complex encryption methods.

#include <iostream>

#include <string>

using namespace std;

// Function to generate the key in a cyclic manner until its length equals the length of the text

string generateKey(string text, string key)

{

    int textLength = text.length();

    int keyLength = key.length();

    for (int i = 0; i < textLength - keyLength; i++)

    {

        key += key[i % keyLength];

    }

    return key;

}

// Function to encrypt the plaintext using Vigenère Cipher

string vigenereEncrypt(string text, string key)

{

    string cipherText = "";

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

    {

        // Convert letters to numbers, A=0, B=1, ..., Z=25

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

        // Convert numbers back to letters

        x += 'A';

        cipherText += x;

    }

    return cipherText;

}

// Function to decrypt the ciphertext using Vigenère Cipher

string vigenereDecrypt(string cipherText, string key)

{

    string plainText = "";

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

    {

        // Convert letters to numbers, A=0, B=1, ..., Z=25

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

        // Convert numbers back to letters

        x += 'A';

        plainText += x;

    }

    return plainText;

}

int main()

{

    string text, keyword;

    cout << "Enter the plaintext (uppercase letters only): ";

    getline(cin, text);

    cout << "Enter the keyword (uppercase letters only): ";

    getline(cin, keyword);

    // Generate the key in a cyclic manner

    string key = generateKey(text, keyword);

    // Encrypt the text

    string cipherText = vigenereEncrypt(text, key);

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

    // Decrypt the text

    string decryptedText = vigenereDecrypt(cipherText, key);

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

    return 0;

}

