## Name: **Ketan Kunkalikar**

## Roll: **21CSE1016**

## Lab Subject: **Security Lab**

## Topic: **Playfair and Hill Ciphers**

## Instructor: **Dr. Meenakshi Panda**

## Date: **18 September 2024**

## **What exactly is the Playfair cipher?**

## In the last class and today's class, we learned about the Playfair cipher, a symmetric encryption technique. It encrypts pairs of letters using a 5x5 matrix generated from a keyword, making it more secure than simple substitution ciphers by encrypting digraphs instead of single letters.

## **What exactly is the Hill cipher?**

## The Hill cipher is polyalphabetic substitution cipher. It uses linear algebra and matrix multiplication to encrypt blocks of letters, providing strong security by encrypting multiple letters at once, making it resistant to frequency analysis attacks. Here, we can encrypt 2, 3 or more blocks of letters at a time using Digraphs, trigraphs and polygraphs.

// 1. Write a program to encrypt and decrypt the message "Meet Me at the Bridge" using Play fair cipher where key is "Your Name".

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

vector<vector<char>> createMatrix(string key) {

    vector<vector<char>> matrix(5, vector<char>(5));

    string alphabet = "abcdefghiklmnopqrstuvwxyz";

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

    key.erase(unique(key.begin(), key.end()), key.end());

    int k = 0;

    for (int i = 0; i < 5; i++) {

        for (int j = 0; j < 5; j++) {

            if (k < key.length()) {

                matrix[i][j] = key[k++];

                alphabet.erase(remove(alphabet.begin(), alphabet.end(), key[k-1]), alphabet.end());

            } else {

                matrix[i][j] = alphabet[0];

                alphabet.erase(alphabet.begin());

            }

        }

    }

    cout << "Matrix:\n";

    for (const auto& row : matrix) {

        for (char c : row) {

            cout << c << ' ';

        }

        cout << '\n';

    }

    return matrix;

}

pair<int, int> findPosition(vector<vector<char>>& matrix, char c) {

    for (int i = 0; i < 5; i++) {

        for (int j = 0; j < 5; j++) {

            if (matrix[i][j] == c) {

                return {i, j};

            }

        }

    }

    return {-1, -1};

}

string playfairEncrypt(string message, vector<vector<char>>& matrix) {

    string result;

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

        char a = message[i];

        char b = (i + 1 < message.length()) ? message[i + 1] : 'x';

        if (a == b) b = 'x';

        pair<int, int> pos1 = findPosition(matrix, a);

        pair<int, int> pos2 = findPosition(matrix, b);

        cout << "Encrypting pair: " << a << b << " -> Positions: (" << pos1.first << "," << pos1.second << ") and (" << pos2.first << "," << pos2.second << ")\n";

        if (pos1.first == pos2.first) {

            result += matrix[pos1.first][(pos1.second + 1) % 5];

            result += matrix[pos2.first][(pos2.second + 1) % 5];

        } else if (pos1.second == pos2.second) {

            result += matrix[(pos1.first + 1) % 5][pos1.second];

            result += matrix[(pos2.first + 1) % 5][pos2.second];

        } else {

            result += matrix[pos1.first][pos2.second];

            result += matrix[pos2.first][pos1.second];

        }

        cout << "Encrypted pair: " << result[result.length() - 2] << result[result.length() - 1] << "\n";

    }

    return result;

}

string playfairDecrypt(string message, vector<vector<char>>& matrix) {

    string result;

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

        char a = message[i];

        char b = message[i + 1];

        pair<int, int> pos1 = findPosition(matrix, a);

        int row1 = pos1.first, col1 = pos1.second;

        pair<int, int> pos2 = findPosition(matrix, b);

        int row2 = pos2.first, col2 = pos2.second;

        cout << "Decrypting pair: " << a << b << " -> Positions: (" << row1 << "," << col1 << ") and (" << row2 << "," << col2 << ")\n";

        if (row1 == row2) {

            result += matrix[row1][(col1 - 1 + 5) % 5];

            result += matrix[row2][(col2 - 1 + 5) % 5];

        } else if (col1 == col2) {

            result += matrix[(row1 - 1 + 5) % 5][col1];

            result += matrix[(row2 - 1 + 5) % 5][col2];

        } else {

            result += matrix[row1][col2];

            result += matrix[row2][col1];

        }

        cout << "Decrypted pair: " << result[result.length() - 2] << result[result.length() - 1] << "\n";

    }

    return result;

}

int main() {

    string key = "ketan";

    string message = "meetmeatthebridge";

    vector<vector<char>> matrix = createMatrix(key);

    string encrypted = playfairEncrypt(message, matrix);

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

    string decrypted = playfairDecrypt(encrypted, matrix);

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

    return 0;

}

## **Output:**

## 

// 2. Write a program to encrypt and decrypt the message "Pay more money" using trigraph Hill Cipher where key is "GYBNQKURP".

#include <iostream>

#include <vector>

#include <string>

using namespace std;

// Function to convert a 3x3 string key to a 3x3 integer matrix

vector<vector<int>> keyToMatrix(const string& key) {

    vector<vector<int>> keyMatrix(3, vector<int>(3));

    for (int i = 0; i < 3; i++) {

        for (int j = 0; j < 3; j++) {

            keyMatrix[i][j] = key[i \* 3 + j] - 'A';

        }

    }

    return keyMatrix;

}

// Function to encrypt a message

string encrypt(const string& message, const vector<vector<int>>& keyMatrix) {

    string encrypted = "";

    for (int i = 0; i < message.length(); i += 3) {

        for (int j = 0; j < 3; j++) {

            int sum = 0;

            for (int k = 0; k < 3; k++) {

                sum += keyMatrix[j][k] \* (message[i + k] - 'A');

            }

            encrypted += (sum % 26) + 'A';

        }

    }

    return encrypted;

}

// Function to calculate the determinant of a 3x3 matrix

int determinant(const vector<vector<int>>& matrix) {

    return (matrix[0][0] \* (matrix[1][1] \* matrix[2][2] - matrix[1][2] \* matrix[2][1]) -

            matrix[0][1] \* (matrix[1][0] \* matrix[2][2] - matrix[1][2] \* matrix[2][0]) +

            matrix[0][2] \* (matrix[1][0] \* matrix[2][1] - matrix[1][1] \* matrix[2][0])) % 26;

}

// Function to calculate the modular multiplicative inverse

int modInverse(int a, int m) {

    a = a % m;

    for (int x = 1; x < m; x++)

        if ((a \* x) % m == 1)

            return x;

    return 1;

}

// Function to calculate the adjugate matrix

vector<vector<int>> adjugate(const vector<vector<int>>& matrix) {

    vector<vector<int>> adj(3, vector<int>(3));

    adj[0][0] = (matrix[1][1] \* matrix[2][2] - matrix[1][2] \* matrix[2][1]) % 26;

    adj[0][1] = (-(matrix[0][1] \* matrix[2][2] - matrix[0][2] \* matrix[2][1]) + 26) % 26;

    adj[0][2] = (matrix[0][1] \* matrix[1][2] - matrix[0][2] \* matrix[1][1]) % 26;

    adj[1][0] = (-(matrix[1][0] \* matrix[2][2] - matrix[1][2] \* matrix[2][0]) + 26) % 26;

    adj[1][1] = (matrix[0][0] \* matrix[2][2] - matrix[0][2] \* matrix[2][0]) % 26;

    adj[1][2] = (-(matrix[0][0] \* matrix[1][2] - matrix[0][2] \* matrix[1][0]) + 26) % 26;

    adj[2][0] = (matrix[1][0] \* matrix[2][1] - matrix[1][1] \* matrix[2][0]) % 26;

    adj[2][1] = (-(matrix[0][0] \* matrix[2][1] - matrix[0][1] \* matrix[2][0]) + 26) % 26;

    adj[2][2] = (matrix[0][0] \* matrix[1][1] - matrix[0][1] \* matrix[1][0]) % 26;

    return adj;

}

// Function to decrypt a message

string decrypt(const string& encrypted, const vector<vector<int>>& keyMatrix) {

    int det = determinant(keyMatrix);

    cout << "Determinant: " << det << endl;

    int detInverse = modInverse(det, 26);

    cout << "Determinant Inverse: " << detInverse << endl;

    vector<vector<int>> adj = adjugate(keyMatrix);

    vector<vector<int>> inverseMatrix(3, vector<int>(3));

    for (int i = 0; i < 3; i++) {

        for (int j = 0; j < 3; j++) {

            inverseMatrix[i][j] = (adj[i][j] \* detInverse) % 26;

        }

    }

    cout << "Inverse Matrix:" << endl;

    for (const auto& row : inverseMatrix) {

        for (int val : row) {

            cout << val << " ";

        }

        cout << endl;

    }

    vector<vector<int>> inverseMatrixMod25(3, vector<int>(3));

    for (int i = 0; i < 3; i++) {

        for (int j = 0; j < 3; j++) {

            inverseMatrixMod25[i][j] = ((inverseMatrix[i][j] % 25) + 25) % 25;

        }

    }

    cout << "Inverse Matrix Mod 25:" << endl;

    for (const auto& row : inverseMatrixMod25) {

        for (int val : row) {

            cout << val << " ";

        }

        cout << endl;

    }

    string decrypted = "";

    for (int i = 0; i < encrypted.length(); i += 3) {

        for (int j = 0; j < 3; j++) {

            int sum = 0;

            for (int k = 0; k < 3; k++) {

                sum += inverseMatrix[j][k] \* (encrypted[i + k] - 'A');

            }

            decrypted += ((sum % 26 + 26) % 26) + 'A';

        }

    }

    return decrypted;

}

int main() {

    string key = "GYBNQKURP";

    string message = "PAYMOREMONEY";

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

    vector<vector<int>> keyMatrix = keyToMatrix(key);

    cout << "Key Matrix:" << endl;

    for (const auto& row : keyMatrix) {

        for (int val : row) {

            cout << val << " ";

        }

        cout << endl;

    }

    string encrypted = encrypt(message, keyMatrix);

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

    string decrypted = decrypt(encrypted, keyMatrix);

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

    return 0;

}

## 