Walchand College of Engineering, Sangli (Government Aided Autonomous Institute)	
AY 2025-26	
Course Information	
Programme	B.Tech. (Computer Science and Engineering)
Class, Semester	Final Year B. Tech., Sem VII
Course Code	6CS451 C
Course Name	Cryptography and Network Security Lab

Experiment No. 01

Name : Soham U. Patil

PRN NO : 22510019

Batch : B6

Title – Encryption and Decryption Using Substitution Techniques

Objectives:

Develop a cryptographic system that implements four classical substitution ciphers for both encryption and decryption:

1. Caesar Cipher
2. Playfair Cipher
3. Hill Cipher
4. Vigenere Cipher

The system should take plaintext input and return the corresponding ciphertext for each cipher method and be able to reverse the process to recover the plaintext from the ciphertext.

Theory:

Substitution Techniques

Substitution ciphers are encryption methods where each element of the plaintext (letters, groups of letters, or bits) is replaced with another element according to a defined mapping. The security of these ciphers relies on the secrecy of the mapping (key).

In classical cryptography, substitution ciphers are simple but form the basis of many modern cryptosystems.

1. Caesar Cipher

- **Definition**: A monoalphabetic cipher where each letter of the plaintext is shifted by a fixed number of positions down the alphabet.

  Caesar Cipher is a substitution cipher where each letter in the plaintext is shifted by a fixed number of positions in the alphabet.

  Encryption Formula:
  C=(P+k)mod 26C = (P + k) \mod 26C=(P+k)mod26

  Decryption Formula:
  P=(C−k)mod 26P = (C - k) \mod 26P=(C−k)mod26

  Here:

  - P = plaintext letter index (A=0, B=1, ..., Z=25)
  - C = ciphertext letter index
  - k = shift key

Example:

- Plaintext: HELLO
- Key (shift): 3

Encryption Steps:

1. Convert to numbers: H=7, E=4, L=11, O=14
   → [7, 4, 11, 11, 14]
2. Apply formula: C=(P+3)mod 26C = (P + 3) \mod 26C=(P+3)mod26
   → [10, 7, 14, 14, 17]
3. Convert back to letters: K, H, O, O, R

Ciphertext: “KHOOR”

Decryption Steps:

1. Convert ciphertext: K=10, H=7, O=14, R=17
   → [10, 7, 14, 14, 17]
2. Apply formula: P=(C−3)mod 26P = (C - 3) \mod 26P=(C−3)mod26
   → [7, 4, 11, 11, 14]
3. Convert back: H, E, L, L, O

Recovered Plaintext: "HELLO"

2. Playfair Cipher

- **Definition**: A digraph substitution cipher invented by Charles Wheatstone, popularized by Lord Playfair. Uses a **5×5 matrix** of letters constructed from a keyword.

Steps :

- Uses a 5×5 matrix built with a keyword (I and J are combined).
- Encrypts pairs of letters (digraphs).
- Rules:
  1. If letters are in the same row → replace with letters to the right.
  2. If letters are in the same column → replace with letters below.
  3. Otherwise → form rectangle and take letters from opposite corners.

Example:

- Keyword: MONARCHY
- Plaintext: HELLO

  Step 1: Create 5×5 Matrix

  M O N A R
  C H Y B D
  E F G I/J K
  L P Q S T
  U V W X Z

  Step 2: Split into pairs
  HELLO → HE | LX | LO (extra X added to avoid duplicate Ls)

  Step 3: Encryption

  - HE → H & E form rectangle → CF
  - LX → L & X form rectangle → SU
  - LO → L & O form rectangle → PM

  Ciphertext: **CFSUPM**

Decryption

- Reverse rules → get back "HELXLO" → remove extra X → HELLO

Recovered Plaintext: HELLO

3. Hill Cipher

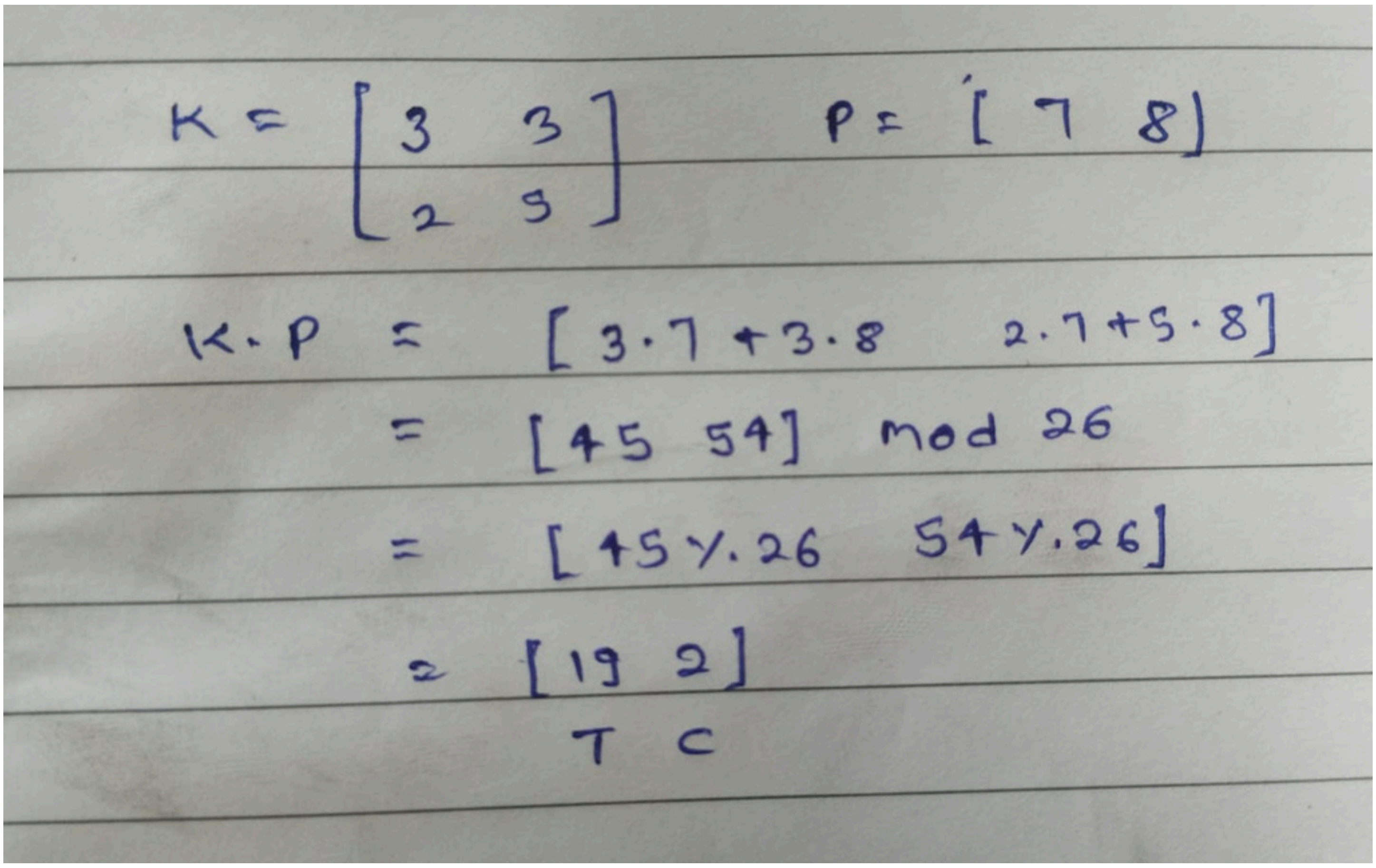
- **Definition**: A polygraphic substitution cipher based on linear algebra.

Theory:

- Based on linear algebra.
- Encryption: $C = K \cdot P \mod 26$
- Decryption: $P = K^{-1} \cdot C \mod 26$
- K = key matrix (must be invertible mod 26).

Example:

- Key matrix:
  K=[3325]K = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix}K=[3235]
- Plaintext: **HI**



$K = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \quad P = [7 \quad 8]$

$K \cdot P = [3 \cdot 7 + 3 \cdot 8 \quad 2 \cdot 7 + 5 \cdot 8]$

$= [45 \quad 54] \bmod 26$

$= [45 \% 26 \quad 54 \% 26]$

$= [19 \quad 2]$

T C

Step 1: Convert letters to numbers
H=7, I=8 → P=[78]P = \begin{bmatrix} 7 \\ 8 \end{bmatrix}P=[78]

Step 2: Encrypt
C=K · Pmod 26C = K \cdot P \mod 26C=K · Pmod26

C=[3325][78]=[4554]C = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 7 \\ 8 \end{bmatrix} = \begin{bmatrix} 45 \\ 54 \end{bmatrix}C=[3235][78]=[4554] Cmod 26=[192]C \mod 26 = \begin{bmatrix} 19 \\ 2 \end{bmatrix}Cmod26=[192]

Numbers → T=19, C=2

Ciphertext: TC

Step 3: Decryption

1. Find K^{-1} C mod 26 (inverse matrix modulo 26).
2. Multiply ciphertext vector with inverse to get plaintext.

Recovered Plaintext: HI

4. Vigenère Cipher

- **Definition**: A polyalphabetic cipher that uses a keyword to shift each letter of the plaintext by varying amounts.

Theory:

- Polyalphabetic cipher (uses a keyword).
- Encryption: Ci=(Pi+Ki)mod 26
- Decryption: Pi=(Ci−Ki)mod 26

Example:

- Plaintext: HELLO
- Keyword: KEY

- Step 1: Repeat keyword
  KEYKE

- Step 2: Convert to numbers
  Plaintext: H=7, E=4, L=11, L=11, O=14
  Keyword: K=10, E=4, Y=24, K=10, E=4

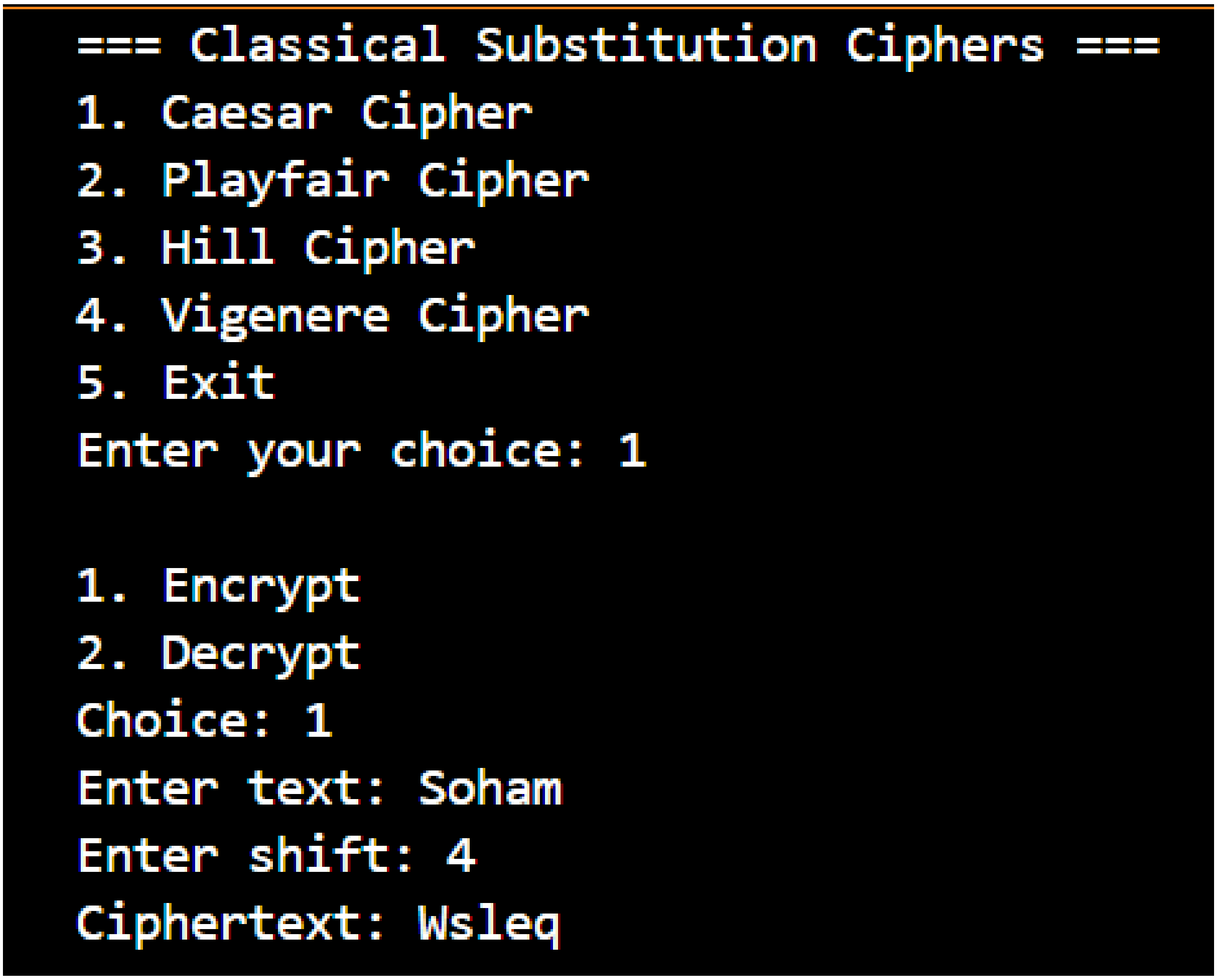
- Step 3: Encrypt
  (7+10)%26=17 → R
  (4+4)%26=8 → I
  (11+24)%26=9 → J
  (11+10)%26=21 → V
  (14+4)%26=18 → S

Ciphertext: RIJVS

- Step 4: Decrypt
  (17−10)%26=7 → H
  (8−4)%26=4 → E
  (9−24)%26=11 → L
  (21−10)%26=11 → L
  (18−4)%26=14 → O

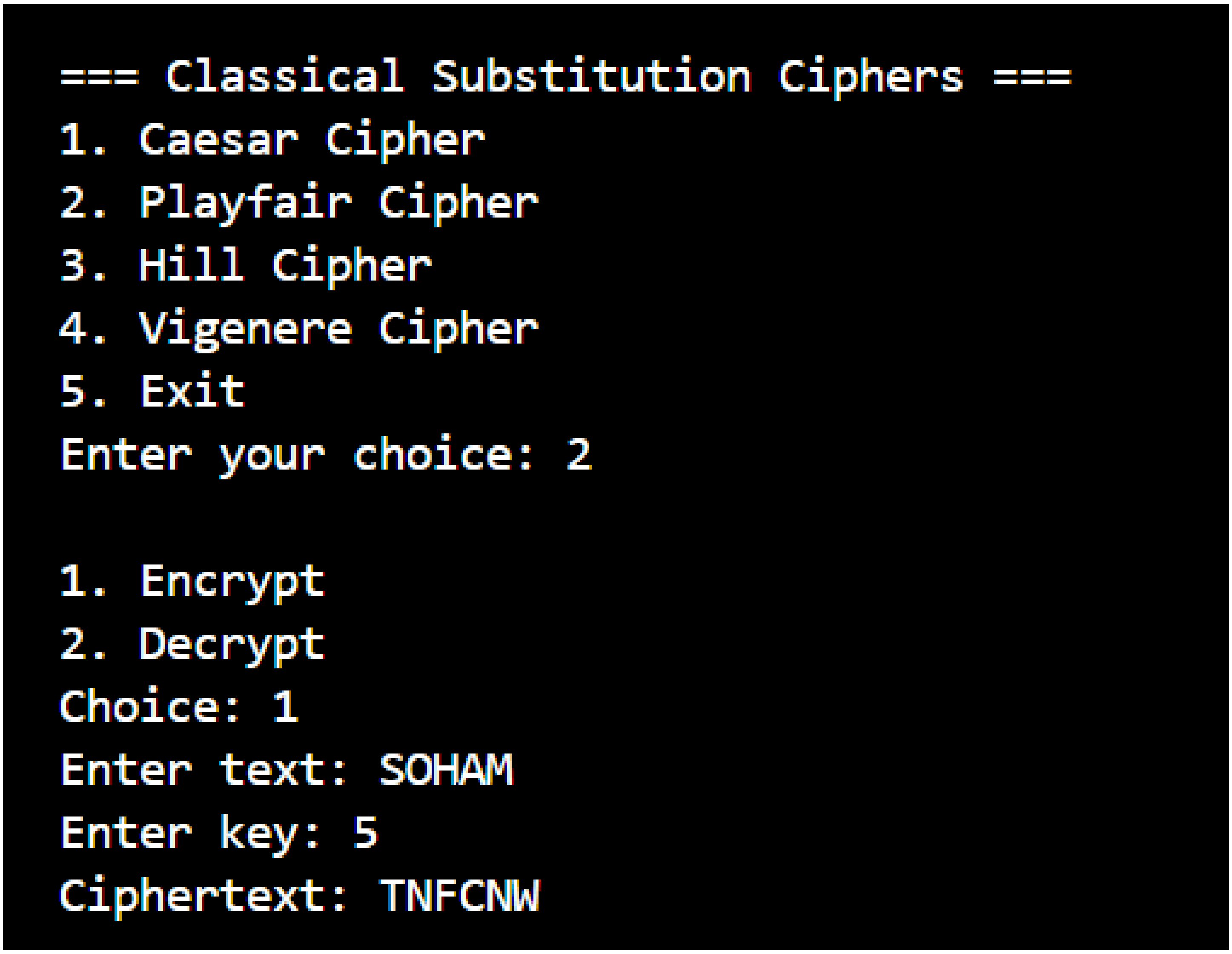
Recovered Plaintext: HELLO

output ;



```
=== Classical Substitution Ciphers ===
1. Caesar Cipher
2. Playfair Cipher
3. Hill Cipher
4. Vigenere Cipher
5. Exit
Enter your choice: 1

1. Encrypt
2. Decrypt
Choice: 1
Enter text: Soham
Enter shift: 4
Ciphertext: Wsleq
```



```
=== Classical Substitution Ciphers ===
1. Caesar Cipher
2. Playfair Cipher
3. Hill Cipher
4. Vigenere Cipher
5. Exit
Enter your choice: 2

1. Encrypt
2. Decrypt
Choice: 1
Enter text: SOHAM
Enter key: 5
Ciphertext: TNFCNW
```

NOTE : Code is attached with the zip file

Code :

```
#include <bits/stdc++.h>

using namespace std;


// ---------- a. Caesar Cipher ----------

string caesarEncrypt(const string &plaintext, int shift) {

    string ciphertext;
```

```cpp
    for (char ch : plaintext) {

        if (isalpha(ch)) {

            char offset = isupper(ch) ? 'A' : 'a';

            ciphertext += char((ch - offset + shift + 26) % 26 + offset);

        } else {

            ciphertext += ch;

        }

    }

    return ciphertext;

}


string caesarDecrypt(const string &ciphertext, int shift) {

    return caesarEncrypt(ciphertext, -shift);

}


// ---------- b. Playfair Cipher ----------

string playfairPrepareText(string text) {

    for (char &c : text) {

        c = toupper(c);

        if (c == 'J') c = 'I';

    }

    string prepared;

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

        char char1 = text[i];

        if (!isalpha(char1)) { i++; continue; }

        if (i + 1 < (int)text.size()) {

            char char2 = text[i + 1];
            if (!isalpha(char2)) { prepared += char1; i++; continue; }
            if (char1 == char2) {
                prepared += char1;
                prepared += 'X';
                i++;
            } else {
                prepared += char1;
                prepared += char2;
                i += 2;
            }
        } else {
            prepared += char1;
            prepared += 'X';
            i++;
        }
    }
    return prepared;
}

vector<vector<char>> playfairGenerateKeyMatrix(string key) {
    for (char &c : key) {
        c = toupper(c);
        if (c == 'J') c = 'I';
    }
    vector<char> matrix;
    set<char> used;
    for (char c : key) {
```

```cpp
        if (isalpha(c) && !used.count(c)) {
            matrix.push_back(c);
            used.insert(c);
        }
    }
    for (char c = 'A'; c <= 'Z'; c++) {
        if (c == 'J') continue;
        if (!used.count(c)) {
            matrix.push_back(c);
            used.insert(c);
        }
    }
    vector<vector<char>> mat(5, vector<char>(5));
    for (int i = 0; i < 25; i++) {
        mat[i / 5][i % 5] = matrix[i];
    }
    return mat;
}

pair<int, int> playfairFindPos(const vector<vector<char>> &matrix, char ch) {
    for (int r = 0; r < 5; r++) {
        for (int c = 0; c < 5; c++) {
            if (matrix[r][c] == ch)
                return make_pair(r, c);
        }
    }
    return make_pair(-1, -1);
```

```
}

string playfairEncrypt(string plaintext, string key) {
    auto matrix = playfairGenerateKeyMatrix(key);
    string prepared = playfairPrepareText(plaintext);
    string ciphertext;
    for (int i = 0; i < (int)prepared.size(); i += 2) {
        char a = prepared[i], b = prepared[i + 1];
        auto pos1 = playfairFindPos(matrix, a);
        auto pos2 = playfairFindPos(matrix, b);
        int r1 = pos1.first, c1 = pos1.second;
        int r2 = pos2.first, c2 = pos2.second;
        if (r1 == r2) {
            ciphertext += matrix[r1][(c1 + 1) % 5];
            ciphertext += matrix[r2][(c2 + 1) % 5];
        } else if (c1 == c2) {
            ciphertext += matrix[(r1 + 1) % 5][c1];
            ciphertext += matrix[(r2 + 1) % 5][c2];
        } else {
            ciphertext += matrix[r1][c2];
            ciphertext += matrix[r2][c1];
        }
    }
    return ciphertext;
}

string playfairDecrypt(string ciphertext, string key) {
    auto matrix = playfairGenerateKeyMatrix(key);

    string plaintext;

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

        char a = ciphertext[i], b = ciphertext[i + 1];

        auto pos1 = playfairFindPos(matrix, a);

        auto pos2 = playfairFindPos(matrix, b);

        int r1 = pos1.first, c1 = pos1.second;

        int r2 = pos2.first, c2 = pos2.second;

        if (r1 == r2) {

            plaintext += matrix[r1][(c1 - 1 + 5) % 5];

            plaintext += matrix[r2][(c2 - 1 + 5) % 5];

        } else if (c1 == c2) {

            plaintext += matrix[(r1 - 1 + 5) % 5][c1];

            plaintext += matrix[(r2 - 1 + 5) % 5][c2];

        } else {

            plaintext += matrix[r1][c2];

            plaintext += matrix[r2][c1];

        }

    }

    return plaintext;

}


// ---------- c. Hill Cipher ----------

int modInverse(int a, int m) {

    a %= m;

    for (int x = 1; x < m; x++)

        if ((a * x) % m == 1) return x;
    return -1;
}

vector<vector<int>> matrixModInverse(vector<vector<int>> m, int mod) {
    int n = m.size();
    int det = round(m[0][0] * m[1][1] - m[0][1] * m[1][0]); // 2x2 case
    det = (det % mod + mod) % mod;
    int detInv = modInverse(det, mod);
    if (detInv == -1) throw runtime_error("Matrix not invertible");

    vector<vector<int>> inv(n, vector<int>(n));
    inv[0][0] = m[1][1];
    inv[1][1] = m[0][0];
    inv[0][1] = -m[0][1];
    inv[1][0] = -m[1][0];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            inv[i][j] = ((inv[i][j] % mod + mod) % mod * detInv) % mod;
    return inv;
}

string hillEncrypt(string plaintext, vector<vector<int>> key) {
    int n = key.size();
    for (char &c : plaintext) c = toupper(c);
    plaintext.erase(remove(plaintext.begin(), plaintext.end(), ' '), plaintext.end());
    while (plaintext.size() % n != 0) plaintext += 'X';
    string ciphertext;
    for (int i = 0; i < (int)plaintext.size(); i += n) {
        vector<int> vec(n);
        for (int j = 0; j < n; j++) vec[j] = plaintext[i + j] - 'A';
        vector<int> res(n, 0);
        for (int r = 0; r < n; r++)
            for (int c = 0; c < n; c++)
                res[r] += key[r][c] * vec[c];
        for (int r = 0; r < n; r++)
            ciphertext += char((res[r] % 26 + 26) % 26 + 'A');
    }
    return ciphertext;
}

string hillDecrypt(string ciphertext, vector<vector<int>> key) {
    int n = key.size();
    auto invKey = matrixModInverse(key, 26);
    string plaintext;
    for (int i = 0; i < (int)ciphertext.size(); i += n) {
        vector<int> vec(n);
        for (int j = 0; j < n; j++) vec[j] = ciphertext[i + j] - 'A';
        vector<int> res(n, 0);
        for (int r = 0; r < n; r++)
            for (int c = 0; c < n; c++)
                res[r] += invKey[r][c] * vec[c];
        for (int r = 0; r < n; r++)
            plaintext += char((res[r] % 26 + 26) % 26 + 'A');
    }
```

```cpp
    return plaintext;
}

// ---------- d. Vigenère Cipher ----------
string vigenereEncrypt(const string &plaintext, string key) {
    for (char &c : key) c = toupper(c);
    string ciphertext;
    int keyLen = key.size();
    for (int i = 0; i < (int)plaintext.size(); i++) {
        char ch = plaintext[i];
        if (isalpha(ch)) {
            char offset = isupper(ch) ? 'A' : 'a';
            int keyVal = key[i % keyLen] - 'A';
            ciphertext += char((ch - offset + keyVal) % 26 + offset);
        } else {
            ciphertext += ch;
        }
    }
    return ciphertext;
}

string vigenereDecrypt(const string &ciphertext, string key) {
    for (char &c : key) c = toupper(c);
    string plaintext;
    int keyLen = key.size();
    for (int i = 0; i < (int)ciphertext.size(); i++) {
        char ch = ciphertext[i];
        if (isalpha(ch)) {
            char offset = isupper(ch) ? 'A' : 'a';
            int keyVal = key[i % keyLen] - 'A';
            plaintext += char((ch - offset - keyVal + 26) % 26 + offset);
        } else {
            plaintext += ch;
        }
    }
    return plaintext;
}

// ---------- Wrapper Functions ----------
void caesarCipher() {
    string text;
    int shift, choice;
    cout << "\n1. Encrypt\n2. Decrypt\nChoice: ";
    cin >> choice;
    cin.ignore();
    cout << "Enter text: ";
    getline(cin, text);
    cout << "Enter shift: ";
    cin >> shift;
    if (choice == 1)
        cout << "Ciphertext: " << caesarEncrypt(text, shift) << endl;
    else
        cout << "Plaintext: " << caesarDecrypt(text, shift) << endl;
}
```

```cpp

void playfairCipher() {

    string text, key;

    int choice;

    cout << "\n1. Encrypt\n2. Decrypt\nChoice: ";

    cin >> choice;

    cin.ignore();

    cout << "Enter text: ";

    getline(cin, text);

    cout << "Enter key: ";

    getline(cin, key);

    if (choice == 1)

        cout << "Ciphertext: " << playfairEncrypt(text, key) << endl;

    else

        cout << "Plaintext: " << playfairDecrypt(text, key) << endl;

}


void hillCipher() {

    string text;

    int choice;

    cout << "\n1. Encrypt\n2. Decrypt\nChoice: ";

    cin >> choice;

    cin.ignore();

    cout << "Enter text (A-Z only): ";

    getline(cin, text);

    vector<vector<int>> key = {{3, 3}, {2, 5}}; // Example 2x2 key

    if (choice == 1)
```

```
        cout << "Ciphertext: " << hillEncrypt(text, key) << endl;
    else
        cout << "Plaintext: " << hillDecrypt(text, key) << endl;
}

void vigenereCipher() {
    string text, key;
    int choice;
    cout << "\n1. Encrypt\n2. Decrypt\nChoice: ";
    cin >> choice;
    cin.ignore();
    cout << "Enter text: ";
    getline(cin, text);
    cout << "Enter key: ";
    getline(cin, key);
    if (choice == 1)
        cout << "Ciphertext: " << vigenereEncrypt(text, key) << endl;
    else
        cout << "Plaintext: " << vigenereDecrypt(text, key) << endl;
}

// ---------- Main ----------
int main() {
    int choice;
    while (true) {
        cout << "\n=== Classical Substitution Ciphers ===\n";
        cout << "1. Caesar Cipher\n";
        cout << "2. Playfair Cipher\n";
        cout << "3. Hill Cipher\n";
        cout << "4. Vigenere Cipher\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1: caesarCipher(); break;
            case 2: playfairCipher(); break;
            case 3: hillCipher(); break;
            case 4: vigenereCipher(); break;
            case 5: cout << "Exiting program...\n"; return 0;
            default: cout << "Invalid choice! Try again.\n";
        }
    }
}
```

Conclusions

- All four classical substitution techniques (Caesar, Playfair, Hill, and Vigenère) were successfully implemented for both encryption and decryption.
- The decrypted text matched the original plaintext for all ciphers, confirming correctness of the algorithms.

- Caesar Cipher is simplest but most vulnerable to brute-force and frequency analysis.
- Playfair Cipher increases security by encrypting digraphs instead of single letters.
- Hill Cipher uses linear algebra, allowing polygraphic encryption, but requires an invertible key matrix.
- Vigenère Cipher offers better resistance against frequency analysis by using polyalphabetic substitution.
- These ciphers demonstrate the fundamental principles of classical cryptography, which form the basis for more secure modern algorithms.