Assignment No. 01

Title - Encryption and Decryption Using Substitution Techniques

---

Objectives:

- To understand the working of different classical substitution ciphers.
- To perform encryption and decryption using:
  - a) Caesar Cipher
  - b) Playfair Cipher
  - c) Hill Cipher
  - d) Vigenère Cipher

Equipment/Tools:

- Python (or any programming language) environment (optional for automated computation)
- Paper and pen (for manual calculations)
- Calculator (for matrix operations in Hill Cipher)

Theory:

a) Caesar Cipher:

- It is a substitution cipher where each letter in the plaintext is shifted by a fixed number of positions down the alphabet.
- Encryption: C=(P+k)mod 26C = (P + k) \mod 26
- Decryption: P=(C−k)mod 26P = (C - k) \mod 26
- where PP is plaintext letter index, CC is ciphertext letter index, and kk is the key (shift).

b) Playfair Cipher:

- Uses a 5x5 matrix of letters constructed using a keyword.
- Encrypts pairs of letters (digraphs).
- Rules:
  - Same row: replace each with the letter to the right.
  - Same column: replace each with the letter below.
  - Rectangle: replace letters with letters on the same row but in the other corners of the rectangle.

c) Hill Cipher:

- Uses linear algebra.
- Encryption: C=KPmod 26C = KP \mod 26
- Decryption: P=K−1Cmod 26P = K^{-1}C \mod 26
- KK is the key matrix.
- Requires invertible key matrix modulo 26.

d) Vigenère Cipher:

- A polyalphabetic substitution cipher using a keyword.
- Encryption: Ci=(Pi+Ki)mod 26C_i = (P_i + K_i) \mod 26
- Decryption: Pi=(Ci−Ki)mod 26P_i = (C_i - K_i) \mod 26
- where PiP_i, CiC_i, and KiK_i are the letter indices of plaintext, ciphertext, and key respectively.

Procedure:

a) Caesar Cipher

Example:

- Plaintext: "HELLO"
- Key (shift): 3

Encryption:

1. Convert letters to numbers (A=0, B=1,..., Z=25).
2. Apply C=(P+3)mod 26C = (P + 3) \mod 26.
3. Convert numbers back to letters.

Decryption:

1. Apply P=(C−3)mod 26P = (C - 3) \mod 26.
2. Convert numbers back to letters.

---

b) Playfair Cipher

Example:

- Keyword: "MONARCHY"
- Plaintext: "HELLO"

Steps:

1. Create 5x5 matrix with keyword letters (no repeats), then fill remaining letters (I/J combined).
2. Split plaintext into pairs: "HE", "LX", "LO" (X added if repeated letters).
3. Apply Playfair rules to encrypt.
4. Decrypt by reversing the process.

---

c) Hill Cipher

Example:

- Key matrix K=[3325]K = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix}
- Plaintext: "HI" → convert to vector P=[78]P = \begin{bmatrix} 7 \\ 8 \end{bmatrix} (H=7, I=8)

Encryption:

1. Calculate C=KPmod 26C = KP \mod 26.
2. Convert ciphertext numbers to letters.

Decryption:

1. Compute K−1K^{-1} modulo 26.
2. Calculate P=K−1Cmod 26P = K^{-1}C \mod 26.
3. Convert to letters.

---

d) Vigenère Cipher

Example:

- Plaintext: "HELLO"
- Keyword: "KEY"

Encryption:

1. Repeat keyword: KEYKE
2. Convert letters to numbers.
3. Ci=(Pi+Ki)mod 26C_i = (P_i + K_i) \mod 26.
4. Convert back to letters.

Decryption:

1. Pi=(Ci−Ki)mod 26P_i = (C_i - K_i) \mod 26.
2. Convert back to letters.

Python Code for Demonstration

```
Lab: Encryption and Decryption of Various Substitution Ciphers

import numpy as np

# ----------- a. Caesar Cipher ------------

def caesar_encrypt(plaintext, shift):
    ciphertext = ""
    for char in plaintext:
        if char.isalpha():
            offset = 65 if char.isupper() else 97
            ciphertext += chr((ord(char) - offset + shift) %
26 + offset)
        else:
            ciphertext += char
    return ciphertext

def caesar_decrypt(ciphertext, shift):
    return caesar_encrypt(ciphertext, -shift)


# ----------- b. Playfair Cipher ------------

def playfair_prepare_text(text):
    text = text.upper().replace('J', 'I')
    prepared = ""
    i = 0
    while i < len(text):
        char1 = text[i]
        if i + 1 < len(text):
            char2 = text[i+1]
            if char1 == char2:
                prepared += char1 + 'X'
                i += 1
            else:
                prepared += char1 + char2
                i += 2
        else:
            prepared += char1 + 'X'
            i += 1
    return prepared

def playfair_generate_key_matrix(key):
    key = key.upper().replace('J', 'I')
    matrix = []
    used = set()
    for char in key:
        if char not in used and char.isalpha():
            matrix.append(char)
```

```
            used.add(char)
    for char in "ABCDEFGHIKLMNOPQRSTUVWXYZ":
        if char not in used:
            matrix.append(char)
            used.add(char)
    return np.array(matrix).reshape(5,5)

def playfair_find_pos(matrix, char):
    pos = np.where(matrix == char)
    return pos[0][0], pos[1][0]

def playfair_encrypt(plaintext, key):
    matrix = playfair_generate_key_matrix(key)
    prepared_text = playfair_prepare_text(plaintext)
    ciphertext = ""
    for i in range(0, len(prepared_text), 2):
        a, b = prepared_text[i], prepared_text[i+1]
        r1, c1 = playfair_find_pos(matrix, a)
        r2, c2 = playfair_find_pos(matrix, b)
        if r1 == r2:
            ciphertext += matrix[r1, (c1+1) % 5] + matrix[r2,
(c2+1) % 5]
        elif c1 == c2:
            ciphertext += matrix[(r1+1) % 5, c1] +
matrix[(r2+1) % 5, c2]
        else:
            ciphertext += matrix[r1, c2] + matrix[r2, c1]
    return ciphertext

def playfair_decrypt(ciphertext, key):
    matrix = playfair_generate_key_matrix(key)
    plaintext = ""
    for i in range(0, len(ciphertext), 2):
        a, b = ciphertext[i], ciphertext[i+1]
        r1, c1 = playfair_find_pos(matrix, a)
        r2, c2 = playfair_find_pos(matrix, b)
        if r1 == r2:
            plaintext += matrix[r1, (c1-1) % 5] + matrix[r2,
(c2-1) % 5]
        elif c1 == c2:
            plaintext += matrix[(r1-1) % 5, c1] + matrix[(r2-
1) % 5, c2]
        else:
            plaintext += matrix[r1, c2] + matrix[r2, c1]
    return plaintext


# ---------- c. Hill Cipher -----------

def hill_encrypt(plaintext, key_matrix):
    n = key_matrix.shape[0]
```

```
    # Prepare text: remove spaces and convert to uppercase
    plaintext = plaintext.upper().replace(" ", "")
    # Pad plaintext if not multiple of n
    while len(plaintext) % n != 0:
        plaintext += 'X'
    ciphertext = ""
    for i in range(0, len(plaintext), n):
        block = plaintext[i:i+n]
        vector = [ord(char) - 65 for char in block]
        encrypted_vector = np.dot(key_matrix, vector) % 26
        ciphertext += "".join(chr(num + 65) for num in
encrypted_vector)
    return ciphertext

def hill_decrypt(ciphertext, key_matrix):
    n = key_matrix.shape[0]
    # Find inverse of key matrix mod 26
    det = int(round(np.linalg.det(key_matrix)))  # determinant
    det_inv = None
    for i in range(26):
        if (det * i) % 26 == 1:
            det_inv = i
            break
    if det_inv is None:
        raise ValueError("Matrix not invertible modulo 26")
    # Matrix of cofactors
    cofactors = np.linalg.inv(key_matrix).T * det
    adjugate = np.round(cofactors).astype(int) % 26
    inv_key = (det_inv * adjugate) % 26

    plaintext = ""
    for i in range(0, len(ciphertext), n):
        block = ciphertext[i:i+n]
        vector = [ord(char) - 65 for char in block]
        decrypted_vector = np.dot(inv_key, vector) % 26
        plaintext += "".join(chr(int(round(num)) + 65) for num
in decrypted_vector)
    return plaintext


# ----------- d. Vigenere Cipher ------------

def vigenere_encrypt(plaintext, key):
    ciphertext = ""
    key = key.upper()
    key_length = len(key)
    for i, char in enumerate(plaintext):
        if char.isalpha():
            offset = 65 if char.isupper() else 97
            key_char = key[i % key_length]
            key_val = ord(key_char) - 65
```

```
            ciphertext += chr((ord(char) - offset + key_val) %
26 + offset)
        else:
            ciphertext += char
    return ciphertext

def vigenere_decrypt(ciphertext, key):
    plaintext = ""
    key = key.upper()
    key_length = len(key)
    for i, char in enumerate(ciphertext):
        if char.isalpha():
            offset = 65 if char.isupper() else 97
            key_char = key[i % key_length]
            key_val = ord(key_char) - 65
            plaintext += chr((ord(char) - offset - key_val) %
26 + offset)
        else:
            plaintext += char
    return plaintext


# ----------- Testing all ciphers ------------

def test_all():
    print("----- Caesar Cipher -----")
    text = "HELLO WORLD"
    shift = 3
    encrypted = caesar_encrypt(text, shift)
    print("Encrypted:", encrypted)
    decrypted = caesar_decrypt(encrypted, shift)
    print("Decrypted:", decrypted)
    print()

    print("----- Playfair Cipher -----")
    text = "HELLO"
    key = "MONARCHY"
    encrypted = playfair_encrypt(text, key)
    print("Encrypted:", encrypted)
    decrypted = playfair_decrypt(encrypted, key)
    print("Decrypted:", decrypted)
    print()

    print("----- Hill Cipher -----")
    text = "HELP"
    key_matrix = np.array([[3, 3], [2, 5]])
    encrypted = hill_encrypt(text, key_matrix)
    print("Encrypted:", encrypted)
    decrypted = hill_decrypt(encrypted, key_matrix)
    print("Decrypted:", decrypted)
    print()
```

```python
    print("----- Vigenere Cipher -----")
    text = "HELLO WORLD"
    key = "KEY"
    encrypted = vigenere_encrypt(text, key)
    print("Encrypted:", encrypted)
    decrypted = vigenere_decrypt(encrypted, key)
    print("Decrypted:", decrypted)


if __name__ == "__main__":
    test_all()
```

```java
import java.util.*;

public class SubstitutionCiphersLab {

    // --- Caesar Cipher ---
    public static String caesarEncrypt(String text, int shift) {
        StringBuilder result = new StringBuilder();
        shift = shift % 26;
        for (char c : text.toUpperCase().toCharArray()) {
            if (c >= 'A' && c <= 'Z') {
                char ch = (char) ((c - 'A' + shift) % 26 + 'A');
                result.append(ch);
            } else {
                result.append(c);
            }
        }
        return result.toString();
    }

    public static String caesarDecrypt(String cipher, int shift) {
        return caesarEncrypt(cipher, 26 - (shift % 26));
    }

    // --- Playfair Cipher ---
    static char[][] playfairMatrix = new char[5][5];

    public static void generatePlayfairMatrix(String key) {
        key = key.toUpperCase().replaceAll("[^A-Z]", "").replace("J", "I");
        LinkedHashSet<Character> set = new LinkedHashSet<>();
        for (char c : key.toCharArray()) set.add(c);
        for (char c = 'A'; c <= 'Z'; c++) {
            if (c != 'J') set.add(c);
        }
        Iterator<Character> it = set.iterator();
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
```

```java
                if (it.hasNext()) playfairMatrix[i][j] = it.next();
            }
        }
    }

    public static String playfairEncrypt(String plaintext, String key) {
        generatePlayfairMatrix(key);
        plaintext = plaintext.toUpperCase().replaceAll("[^A-Z]", "").replace("J", "I");
        StringBuilder sb = new StringBuilder();

        // Prepare digraphs
        List<String> digraphs = new ArrayList<>();
        for (int i = 0; i < plaintext.length(); i += 2) {
            char first = plaintext.charAt(i);
            char second = (i + 1) < plaintext.length() ? plaintext.charAt(i + 1) : 'X';
            if (first == second) second = 'X';
            digraphs.add("" + first + second);
            if (first == second) i--;
        }

        for (String pair : digraphs) {
            int[] pos1 = findPosition(pair.charAt(0));
            int[] pos2 = findPosition(pair.charAt(1));

            if (pos1[0] == pos2[0]) {
                // same row
                sb.append(playfairMatrix[pos1[0]][(pos1[1] + 1) % 5]);
                sb.append(playfairMatrix[pos2[0]][(pos2[1] + 1) % 5]);
            } else if (pos1[1] == pos2[1]) {
                // same column
                sb.append(playfairMatrix[(pos1[0] + 1) % 5][pos1[1]]);
                sb.append(playfairMatrix[(pos2[0] + 1) % 5][pos2[1]]);
            } else {
                // rectangle swap columns
                sb.append(playfairMatrix[pos1[0]][pos2[1]]);
                sb.append(playfairMatrix[pos2[0]][pos1[1]]);
            }
        }
        return sb.toString();
    }

    public static String playfairDecrypt(String cipher, String key) {
        generatePlayfairMatrix(key);
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < cipher.length(); i += 2) {
            char first = cipher.charAt(i);
            char second = cipher.charAt(i + 1);
            int[] pos1 = findPosition(first);
            int[] pos2 = findPosition(second);
```

```java
        if (pos1[0] == pos2[0]) {
            sb.append(playfairMatrix[pos1[0]][(pos1[1] + 4) % 5]);
            sb.append(playfairMatrix[pos2[0]][(pos2[1] + 4) % 5]);
        } else if (pos1[1] == pos2[1]) {
            sb.append(playfairMatrix[(pos1[0] + 4) % 5][pos1[1]]);
            sb.append(playfairMatrix[(pos2[0] + 4) % 5][pos2[1]]);
        } else {
            sb.append(playfairMatrix[pos1[0]][pos2[1]]);
            sb.append(playfairMatrix[pos2[0]][pos1[1]]);
        }
    }
    return sb.toString();
}

private static int[] findPosition(char c) {
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            if (playfairMatrix[i][j] == c) return new int[]{i, j};
        }
    }
    return null;
}

// --- Hill Cipher (2x2) ---
private static int mod26(int x) {
    x %= 26;
    return x < 0 ? x + 26 : x;
}

public static String hillEncrypt(String plaintext, int[][] key) {
    plaintext = plaintext.toUpperCase().replaceAll("[^A-Z]", "");
    if (plaintext.length() % 2 != 0) plaintext += "X";

    StringBuilder cipher = new StringBuilder();
    for (int i = 0; i < plaintext.length(); i += 2) {
        int[] vector = {plaintext.charAt(i) - 'A', plaintext.charAt(i + 1) - 'A'};
        int c1 = mod26(key[0][0] * vector[0] + key[0][1] * vector[1]);
        int c2 = mod26(key[1][0] * vector[0] + key[1][1] * vector[1]);
        cipher.append((char) (c1 + 'A'));
        cipher.append((char) (c2 + 'A'));
    }
    return cipher.toString();
}

public static String hillDecrypt(String cipher, int[][] key) {
    // Find inverse of key matrix modulo 26
    int det = mod26(key[0][0] * key[1][1] - key[0][1] * key[1][0]);
    int detInv = modInverse(det, 26);
    if (detInv == -1) return "Inverse doesn't exist, decryption impossible.";
```

```java
    int[][] invKey = new int[2][2];
    invKey[0][0] = mod26(detInv * key[1][1]);
    invKey[0][1] = mod26(-detInv * key[0][1]);
    invKey[1][0] = mod26(-detInv * key[1][0]);
    invKey[1][1] = mod26(detInv * key[0][0]);

    StringBuilder plaintext = new StringBuilder();
    for (int i = 0; i < cipher.length(); i += 2) {
        int[] vector = {cipher.charAt(i) - 'A', cipher.charAt(i + 1) - 'A'};
        int p1 = mod26(invKey[0][0] * vector[0] + invKey[0][1] * vector[1]);
        int p2 = mod26(invKey[1][0] * vector[0] + invKey[1][1] * vector[1]);
        plaintext.append((char) (p1 + 'A'));
        plaintext.append((char) (p2 + 'A'));
    }
    return plaintext.toString();
}

private static 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;
}

// --- Vigenere Cipher ---
public static String vigenereEncrypt(String text, String key) {
    text = text.toUpperCase().replaceAll("[^A-Z]", "");
    key = key.toUpperCase().replaceAll("[^A-Z]", "");
    StringBuilder result = new StringBuilder();
    for (int i = 0, j = 0; i < text.length(); i++) {
        char c = text.charAt(i);
        int shift = key.charAt(j) - 'A';
        char encrypted = (char) ((c - 'A' + shift) % 26 + 'A');
        result.append(encrypted);
        j = (j + 1) % key.length();
    }
    return result.toString();
}

public static String vigenereDecrypt(String cipher, String key) {
    cipher = cipher.toUpperCase().replaceAll("[^A-Z]", "");
    key = key.toUpperCase().replaceAll("[^A-Z]", "");
    StringBuilder result = new StringBuilder();
    for (int i = 0, j = 0; i < cipher.length(); i++) {
        char c = cipher.charAt(i);
        int shift = key.charAt(j) - 'A';
        char decrypted = (char) ((c - 'A' - shift + 26) % 26 + 'A');
        result.append(decrypted);
        j = (j + 1) % key.length();
```

```java
        }
        return result.toString();
    }

    // --- Main method for demonstration ---
    public static void main(String[] args) {
        // Caesar Cipher
        String caesarText = "HELLO WORLD";
        int caesarShift = 3;
        String caesarEncrypted = caesarEncrypt(caesarText, caesarShift);
        String caesarDecrypted = caesarDecrypt(caesarEncrypted, caesarShift);
        System.out.println("Caesar Cipher:");
        System.out.println("Encrypted: " + caesarEncrypted);
        System.out.println("Decrypted: " + caesarDecrypted);
        System.out.println();

        // Playfair Cipher
        String playfairText = "HELLO";
        String playfairKey = "MONARCHY";
        String playfairEncrypted = playfairEncrypt(playfairText, playfairKey);
        String playfairDecrypted = playfairDecrypt(playfairEncrypted, playfairKey);
        System.out.println("Playfair Cipher:");
        System.out.println("Encrypted: " + playfairEncrypted);
        System.out.println("Decrypted: " + playfairDecrypted);
        System.out.println();

        // Hill Cipher
        String hillText = "HELLO";
        int[][] hillKey = { {3, 3}, {2, 5} };
        String hillEncrypted = hillEncrypt(hillText, hillKey);
        String hillDecrypted = hillDecrypt(hillEncrypted, hillKey);
        System.out.println("Hill Cipher:");
        System.out.println("Encrypted: " + hillEncrypted);
        System.out.println("Decrypted: " + hillDecrypted);
        System.out.println();

        // Vigenere Cipher
        String vigenereText = "HELLO";
        String vigenereKey = "KEY";
        String vigenereEncrypted = vigenereEncrypt(vigenereText, vigenereKey);
        String vigenereDecrypted = vigenereDecrypt(vigenereEncrypted, vigenereKey);
        System.out.println("Vigenere Cipher:");
        System.out.println("Encrypted: " + vigenereEncrypted);
        System.out.println("Decrypted: " + vigenereDecrypted);
    }
}
```

Observations and Conclusion:

- Note how the ciphertext changes with each method.
- Understand the strength and weaknesses of each cipher.
- Classical ciphers like Caesar are simple but insecure.
- Polygraphic and polyalphabetic ciphers like Hill and Vigenère are more secure.