PRN – 22510064

PARSHWA HERWADE

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.

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.

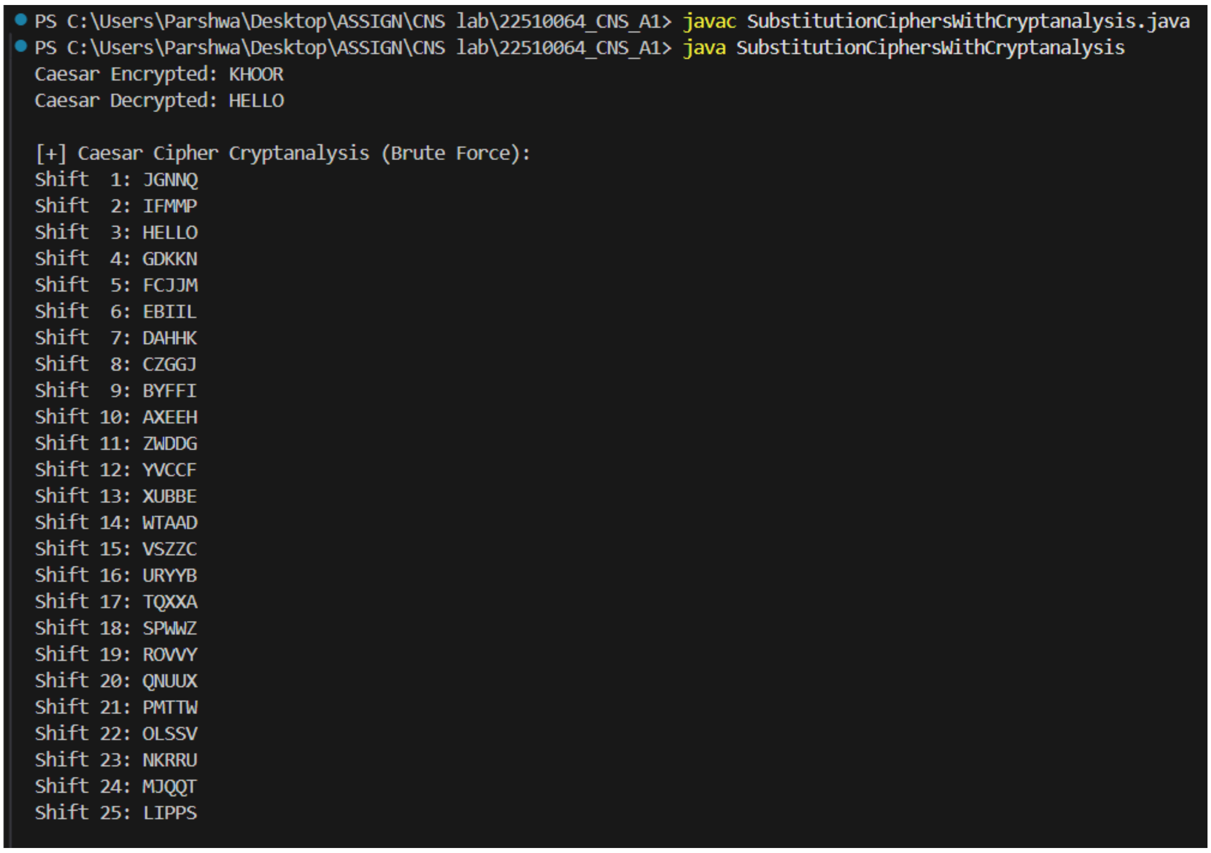
```java
import java.util.*;

public class SubstitutionCiphersWithCryptanalysis {

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

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

    public static void caesarCryptanalysis(String cipher) {
        System.out.println("\n[+] Caesar Cipher Cryptanalysis (Brute
Force):");
        for (int i = 1; i < 26; i++) {
            String attempt = caesarDecrypt(cipher, i);
            System.out.printf("Shift %2d: %s\n", i, attempt);
        }
    }

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

    public static void generatePlayfairMatrix(String key) {
        key = key.toUpperCase().replace("J", "I").replaceAll("[^A-
Z]", "");
        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++)
playfairMatrix[i][j] = it.next();
    }

    private static int[] pos(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;
    }

    public static String playfairEncrypt(String text, String key) {
        generatePlayfairMatrix(key);
        text = text.toUpperCase().replace("J", "I").replaceAll("[^A-
Z]", "");
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < text.length(); i += 2) {
            char a = text.charAt(i);
            char b = (i + 1 < text.length()) ? text.charAt(i + 1) :
'X';
            if (a == b) b = 'X';
            int[] p1 = pos(a), p2 = pos(b);
            if (p1[0] == p2[0]) {
                sb.append(playfairMatrix[p1[0]][(p1[1] + 1) % 5]);
                sb.append(playfairMatrix[p2[0]][(p2[1] + 1) % 5]);
            } else if (p1[1] == p2[1]) {
                sb.append(playfairMatrix[(p1[0] + 1) % 5][p1[1]]);
                sb.append(playfairMatrix[(p2[0] + 1) % 5][p2[1]]);
            } else {
                sb.append(playfairMatrix[p1[0]][p2[1]]);
                sb.append(playfairMatrix[p2[0]][p1[1]]);
            }
        }
        return sb.toString();
    }
```

```java
    public static String playfairDecrypt(String cipher, String key)
{
        generatePlayfairMatrix(key);
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < cipher.length(); i += 2) {
            char a = cipher.charAt(i);
            char b = cipher.charAt(i + 1);
            int[] p1 = pos(a), p2 = pos(b);
            if (p1[0] == p2[0]) {
                sb.append(playfairMatrix[p1[0]][(p1[1] + 4) % 5]);
                sb.append(playfairMatrix[p2[0]][(p2[1] + 4) % 5]);
            } else if (p1[1] == p2[1]) {
                sb.append(playfairMatrix[(p1[0] + 4) % 5][p1[1]]);
                sb.append(playfairMatrix[(p2[0] + 4) % 5][p2[1]]);
            } else {
                sb.append(playfairMatrix[p1[0]][p2[1]]);
                sb.append(playfairMatrix[p2[0]][p1[1]]);
            }
        }
        return sb.toString();
    }

    public static void playfairCryptanalysis(String cipher) {
        System.out.println("\n[+] Playfair Cipher Cryptanalysis
(Simulated):");
        String[] keys = {"MONARCHY", "KEYWORD", "HELLO", "SECRET"};
        for (String k : keys) {
            String plain = playfairDecrypt(cipher, k);
            System.out.println("Trying key \"" + k + "\": " +
plain);
        }
    }

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

    static int modInverse(int a) {
        for (int i = 1; i < 26; i++) if ((a * i) % 26 == 1) return
i;
```

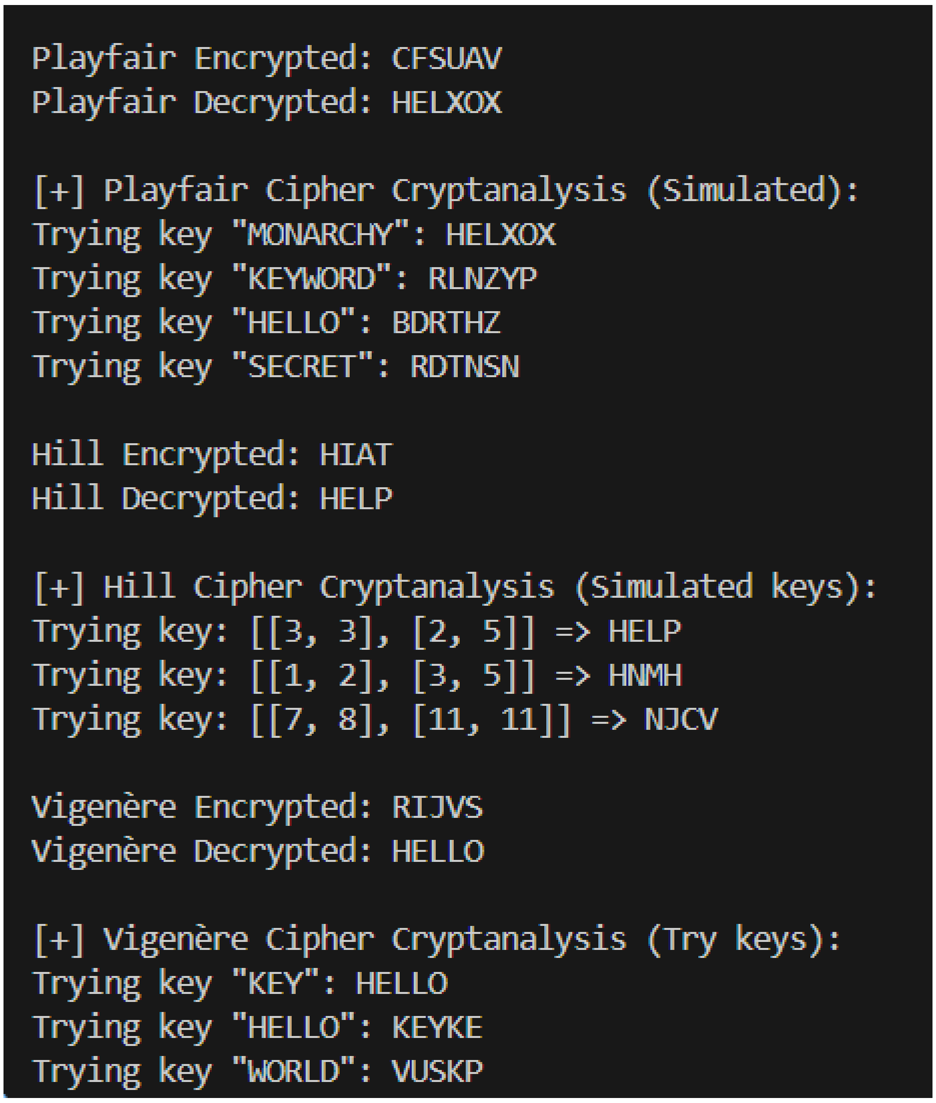
```
        return -1;
    }

    public static String hillEncrypt(String text, int[][] key) {
        text = text.toUpperCase().replaceAll("[^A-Z]", "");
        if (text.length() % 2 != 0) text += "X";
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < text.length(); i += 2) {
            int[] vec = {text.charAt(i) - 'A', text.charAt(i + 1) -
'A'};
            int c1 = mod26(key[0][0] * vec[0] + key[0][1] * vec[1]);
            int c2 = mod26(key[1][0] * vec[0] + key[1][1] * vec[1]);
            sb.append((char) (c1 + 'A')).append((char) (c2 + 'A'));
        }
        return sb.toString();
    }

    public static String hillDecrypt(String cipher, int[][] key) {
        int det = mod26(key[0][0] * key[1][1] - key[0][1] *
key[1][0]);
        int inv = modInverse(det);
        if (inv == -1) return "Matrix not invertible.";
        int[][] invKey = {
                {mod26(inv * key[1][1]), mod26(-inv * key[0][1])},
                {mod26(-inv * key[1][0]), mod26(inv * key[0][0])}
        };
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < cipher.length(); i += 2) {
            int[] vec = {cipher.charAt(i) - 'A', cipher.charAt(i +
1) - 'A'};
            int p1 = mod26(invKey[0][0] * vec[0] + invKey[0][1] *
vec[1]);
            int p2 = mod26(invKey[1][0] * vec[0] + invKey[1][1] *
vec[1]);
            sb.append((char) (p1 + 'A')).append((char) (p2 + 'A'));
        }
        return sb.toString();
    }

    public static void hillCryptanalysis(String cipher) {
        System.out.println("\n[+] Hill Cipher Cryptanalysis
(Simulated keys):");
```

```java
        int[][][] keys = {
                {{3, 3}, {2, 5}}, {{1, 2}, {3, 5}}, {{7, 8}, {11,
11}}
        };
        for (int[][] k : keys) {
            String attempt = hillDecrypt(cipher, k);
            System.out.println("Trying key: " +
Arrays.deepToString(k) + " => " + attempt);
        }
    }

    // === Vigenere Cipher ===
    public static String vigenereEncrypt(String text, String key) {
        text = text.toUpperCase().replaceAll("[^A-Z]", "");
        key = key.toUpperCase();
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < text.length(); i++) {
            int t = text.charAt(i) - 'A';
            int k = key.charAt(i % key.length()) - 'A';
            sb.append((char) ((t + k) % 26 + 'A'));
        }
        return sb.toString();
    }

    public static String vigenereDecrypt(String cipher, String key)
{
        cipher = cipher.toUpperCase();
        key = key.toUpperCase();
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < cipher.length(); i++) {
            int c = cipher.charAt(i) - 'A';
            int k = key.charAt(i % key.length()) - 'A';
            sb.append((char) ((c - k + 26) % 26 + 'A'));
        }
        return sb.toString();
    }

    public static void vigenereCryptanalysis(String cipher) {
        System.out.println("\n[+] Vigenère Cipher Cryptanalysis (Try
keys):");
        String[] keys = {"KEY", "HELLO", "WORLD"};
        for (String k : keys) {
            String attempt = vigenereDecrypt(cipher, k);
            System.out.println("Trying key \"" + k + "\": " +
attempt);
        }
    }

    // === Main ===
    public static void main(String[] args) {
        String plain = "HELLO";

        // Caesar
        String caesarEnc = caesarEncrypt(plain, 3);
        System.out.println("Caesar Encrypted: " + caesarEnc);
        System.out.println("Caesar Decrypted: " +
caesarDecrypt(caesarEnc, 3));
        caesarCryptanalysis(caesarEnc);

        // Playfair
        String playfairEnc = playfairEncrypt(plain, "MONARCHY");
        System.out.println("\nPlayfair Encrypted: " + playfairEnc);
        System.out.println("Playfair Decrypted: " +
playfairDecrypt(playfairEnc, "MONARCHY"));
        playfairCryptanalysis(playfairEnc);

        // Hill
        int[][] hillKey = {{3, 3}, {2, 5}};
        String hillEnc = hillEncrypt("HELP", hillKey);
        System.out.println("\nHill Encrypted: " + hillEnc);
        System.out.println("Hill Decrypted: " + hillDecrypt(hillEnc,
hillKey));
        hillCryptanalysis(hillEnc);

        // Vigenere
        String vigenereEnc = vigenereEncrypt(plain, "KEY");
        System.out.println("\nVigenère Encrypted: " + vigenereEnc);
        System.out.println("Vigenère Decrypted: " +
vigenereDecrypt(vigenereEnc, "KEY"));
        vigenereCryptanalysis(vigenereEnc);
    }
}
```

OUTPUT:



```
PS C:\Users\Parshwa\Desktop\ASSIGN\CNS lab\22510064_CNS_A1> javac SubstitutionCiphersWithCryptanalysis.java
PS C:\Users\Parshwa\Desktop\ASSIGN\CNS lab\22510064_CNS_A1> java SubstitutionCiphersWithCryptanalysis
Caesar Encrypted: KHOOR
Caesar Decrypted: HELLO

[+] Caesar Cipher Cryptanalysis (Brute Force):
Shift  1: JGNNQ
Shift  2: IFMMP
Shift  3: HELLO
Shift  4: GDKKN
Shift  5: FCJJM
Shift  6: EBIIL
Shift  7: DAHHK
Shift  8: CZGGJ
Shift  9: BYFFI
Shift 10: AXEEH
Shift 11: ZWDDG
Shift 12: YVCCF
Shift 13: XUBBE
Shift 14: WTAAD
Shift 15: VSZZC
Shift 16: URYYB
Shift 17: TQXXA
Shift 18: SPWWZ
Shift 19: ROVVY
Shift 20: QNUUX
Shift 21: PMTTW
Shift 22: OLSSV
Shift 23: NKRRU
Shift 24: MJQQT
Shift 25: LIPPS
```



```
Playfair Encrypted: CFSUAV
Playfair Decrypted: HELXOX

[+] Playfair Cipher Cryptanalysis (Simulated):
Trying key "MONARCHY": HELXOX
Trying key "KEYWORD": RLNZYP
Trying key "HELLO": BDRTHZ
Trying key "SECRET": RDTNSN

Hill Encrypted: HIAT
Hill Decrypted: HELP

[+] Hill Cipher Cryptanalysis (Simulated keys):
Trying key: [[3, 3], [2, 5]] => HELP
Trying key: [[1, 2], [3, 5]] => HNMH
Trying key: [[7, 8], [11, 11]] => NJCV

Vigenère Encrypted: RIJVS
Vigenère Decrypted: HELLO

[+] Vigenère Cipher Cryptanalysis (Try keys):
Trying key "KEY": HELLO
Trying key "HELLO": KEYKE
Trying key "WORLD": VUSKP
```

OBSERVATIONS:

During the experiment, the behavior and effectiveness of four classical substitution ciphers—Caesar, Playfair, Hill, and Vigenère—were observed through encryption, decryption, and basic cryptanalysis.

Caesar Cipher

The ciphertext obtained was KHOOR and decrypted successfully back to HELLO.

Brute-force cryptanalysis showed that the correct shift value (3) easily revealed the original message.

Observation: The ciphertext shifts all characters uniformly, making patterns obvious.

Conclusion: Caesar Cipher is extremely simple but highly insecure. It can be broken easily with brute-force or frequency analysis since there are only 25 possible key shifts.

Playfair Cipher

The encrypted message was CFSUAV and decrypted to HELXOX (with 'X' padding due to Playfair rules).

Cryptanalysis using different trial keys showed that only the correct key (MONARCHY) revealed a readable message.

Observation: This cipher encrypts letter pairs (digraphs), which reduces the effectiveness of single-letter frequency analysis.

Conclusion: Playfair is more secure than Caesar due to digraph substitution, but still vulnerable to digraph frequency attacks or known plaintext attacks.

Hill Cipher

Encryption of the plaintext gave HIAT, and decryption using matrix [[3,3],[2,5]] correctly returned HELP.

Other simulated key matrices during cryptanalysis produced unreadable text.

Observation: The cipher depends on linear algebra, requiring the key matrix to be invertible modulo 26.

Conclusion: Hill Cipher is stronger than monoalphabetic ciphers due to its polygraphic nature, but can be broken using known plaintext attacks if the key matrix is not kept secret.

Vigenère Cipher

The ciphertext obtained was RIJVS and correctly decrypted back to HELLO using the key KEY.

Attempts with incorrect keys like WORLD failed to produce meaningful plaintext.

Observation: This cipher applies a series of Caesar shifts determined by the keyword, making it much harder to detect frequency patterns.

Conclusion: Vigenère Cipher is a strong polyalphabetic cipher, and among classical methods, it offers the best security—provided the key is sufficiently long and non-repeating.

Final Summary

The ciphertext changed significantly across methods, with increasing complexity and security from Caesar to Vigenère.

Caesar and Playfair are relatively weak and easily broken with simple cryptanalysis.

Hill and Vigenère ciphers are more resistant due to their use of matrices and key-based variable shifting.

Among classical techniques:

Caesar: Basic and insecure.

Playfair: Better but still vulnerable.

Hill: Secure if matrix and plaintext length are well chosen.

Vigenère: Most secure; ideal for understanding polyalphabetic substitution.

Conclusion: While these ciphers are not secure by today's standards, they are essential in understanding how modern cryptography evolved. They highlight the importance of key complexity, randomness, and resistance to frequency analysis in designing secure encryption systems.