**Practical No: 2**

**Practical Name: Study and Implement a program for 5x5 Playfair Cipher to encrypt and decrypt the message.**

**Roll No: 22BCP250**

**Name: Anusi Patel**

**Division and batch: Div4 G7**

**Original Approach:**

**Introduction:** The Playfair cipher is a manual encryption technique and was the first digraph substitution cipher. It was invented in 1854 by Sir Charles Wheatstone, but it was named after Lord Playfair, who promoted its use. The cipher encrypts pairs of letters (digraphs) instead of single letters, which makes it more secure than simple substitution ciphers like the Caesar cipher.

**How the Playfair Cipher Works:**

1. **Preparation**:
   * A 5x5 grid (matrix) is created using a keyword or phrase. The keyword is written first in the grid, without repeating any letters, and then the remaining letters of the alphabet are added in order. Typically, 'I' and 'J' are treated as the same letter to fit the 25-square grid.
2. **Encryption Process**:
   * **Pairing**: The plaintext is divided into digraphs (pairs of two letters). If a pair consists of the same letter (e.g., "LL"), a filler letter like "X" is inserted between them. If the plaintext has an odd number of characters, an extra filler letter is added at the end.
   * **Substitution**: The digraphs are encrypted based on their position in the grid:
     + **Same Row**: If both letters are in the same row, each letter is substituted with the letter immediately to its right (circularly, wrapping back to the start of the row if needed).
     + **Same Column**: If both letters are in the same column, each letter is substituted with the letter immediately below it (circularly, wrapping back to the top of the column if needed).
     + **Rectangle**: If the letters form a rectangle, each letter is replaced by the letter on the same row but at the rectangle's opposite corner.
3. **Decryption**:
   * The decryption process is the reverse of encryption, following similar rules but moving to the left for rows and upwards for columns.

**Example:**

**Steps to Encrypt Using the Playfair Cipher:**

Let's use the keyword **"BATTLE"** to encrypt the phrase "ATTACK AT DAWN" using the Playfair cipher.

* **Step 1: Create the 5x5 Grid**

First, we create the 5x5 grid using the keyword "BATTLE." We remove duplicate letters from the keyword and then fill in the remaining letters of the alphabet. We'll combine 'I' and 'J' into one cell to fit all the letters.

**Keyword:** BATTLE  
**Grid:**

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

* **Step 2: Prepare the Plaintext**

We split "ATTACK AT DAWN" into digraphs (pairs of letters). If any pairs consist of the same letter, we insert a filler letter (usually 'X'). If there's an odd number of letters, we add a filler letter at the end.

**Plaintext:** ATTACK AT DAWN  
**Digraphs:** AT TA CK AT DA WN

* **Step 3: Encrypt the Digraphs**

Now, we encrypt each pair using the Playfair cipher rules:

1. **AT:**

* A (Row 1, Column 2), T (Row 1, Column 3) are in the same row.
* Move each letter to the right: "AT" becomes "TL."

1. **TA:**

* T (Row 1, Column 3), A (Row 1, Column 2) are in the same row.
* Move each letter to the right: "TA" becomes "LT."

1. **CK:**

* C (Row 2, Column 1), K (Row 3, Column 2) form a rectangle.
* Swap corners: "CK" becomes "DM."

1. **AT:**

* Same as the first pair.
* "AT" becomes "TL."

1. **DA:**

* D (Row 2, Column 2), A (Row 1, Column 2) are in the same column.
* Move each letter down: "DA" becomes "KA."

1. **WN:**

* W (Row 5, Column 2), N (Row 3, Column 4) form a rectangle.
* Swap corners: "WN" becomes "YO."
* **Step 4: Combine the Ciphertext**

After encrypting all digraphs, we combine them to form the final ciphertext.

**Ciphertext:** **TL LT DM TL KA YO  
Steps to Decrypt Using the Playfair Cipher:**

1. **Prepare the 5x5 Grid:**
   * Use the same grid that was used for encryption, constructed from the same keyword or phrase.
2. **Break Ciphertext into Digraphs:**
   * Divide the ciphertext into pairs of letters (digraphs). If the ciphertext has an odd number of characters, an additional filler letter may need to be added to make a complete pair.
3. **Decrypt Each Digraph:**
   * For each digraph:
     + **Same Row:** If both letters are in the same row of the grid, replace each letter with the letter immediately to its left (wrap around to the end of the row if necessary).
     + **Same Column:** If both letters are in the same column of the grid, replace each letter with the letter immediately above (wrap around to the bottom of the column if necessary).
     + **Rectangle Formation:** If the letters form a rectangle, replace each letter with the letter in the same row but at the opposite corner of the rectangle.
4. **Combine the Decrypted Digraphs:**
   * Once all digraphs have been decrypted, combine them to form the plaintext.
5. **Remove Filler Letters (if any):**
   * If filler letters were added during encryption to handle duplicate letters or odd-length plaintexts, remove them to reveal the original message.

**Source Code:**

def create\_matrix(key):

matrix = []

alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ"

for char in key:

if char not in matrix and char in alphabet:

matrix.append(char)

for char in alphabet:

if char not in matrix:

matrix.append(char)

return [matrix[i:i + 5] for i in range(0, 25, 5)]

def find\_position(matrix, char):

for i in range(5):

for j in range(5):

if matrix[i][j] == char:

return i, j

return None

def encrypt(plaintext, key):

matrix = create\_matrix(key)

plaintext = plaintext.upper().replace("J", "I")

prepared\_text = ""

i = 0

while i < len(plaintext):

if plaintext[i] < 'A' or plaintext[i] > 'Z':

i += 1

continue

if i + 1 < len(plaintext) and plaintext[i] == plaintext[i + 1]:

prepared\_text += plaintext[i] + 'X'

i += 1

else:

if i + 1 < len(plaintext):

prepared\_text += plaintext[i] + plaintext[i + 1]

i += 2

else:

prepared\_text += plaintext[i] + 'X'

i += 1

ciphertext = ""

for i in range(0, len(prepared\_text), 2):

row1, col1 = find\_position(matrix, prepared\_text[i])

row2, col2 = find\_position(matrix, prepared\_text[i + 1])

if row1 == row2:

ciphertext += matrix[row1][(col1 + 1) % 5]

ciphertext += matrix[row2][(col2 + 1) % 5]

elif col1 == col2:

ciphertext += matrix[(row1 + 1) % 5][col1]

ciphertext += matrix[(row2 + 1) % 5][col2]

else:

ciphertext += matrix[row1][col2]

ciphertext += matrix[row2][col1]

return ciphertext

def decrypt(ciphertext, key):

matrix = create\_matrix(key)

prepared\_text = ""

for i in range(0, len(ciphertext), 2):

row1, col1 = find\_position(matrix, ciphertext[i])

row2, col2 = find\_position(matrix, ciphertext[i + 1])

if row1 == row2: # Same row

prepared\_text += matrix[row1][(col1 - 1) % 5]

prepared\_text += matrix[row2][(col2 - 1) % 5]

elif col1 == col2: # Same column

prepared\_text += matrix[(row1 - 1) % 5][col1]

prepared\_text += matrix[(row2 - 1) % 5][col2]

else: # Rectangle

prepared\_text += matrix[row1][col2]

prepared\_text += matrix[row2][col1]

return prepared\_text

key = "BATTLE"

plaintext = input("Enter the plain text: ")

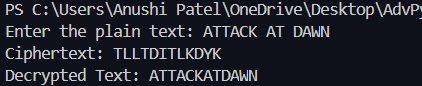
ciphertext = encrypt(plaintext, key)

decrypted\_text = decrypt(ciphertext, key)

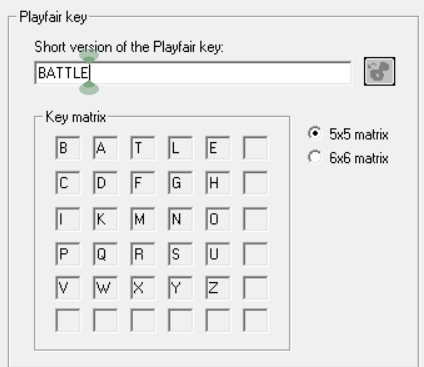
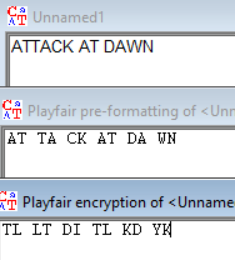
print("Ciphertext:", ciphertext)

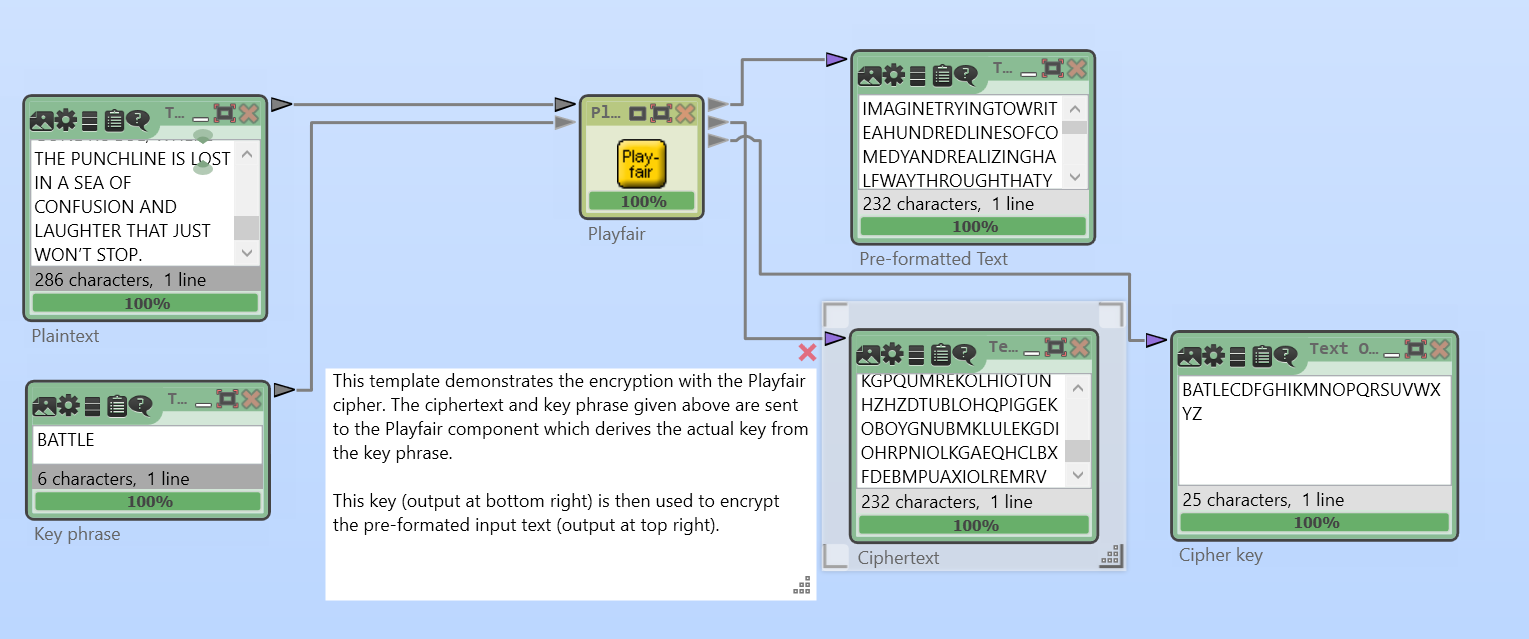
print("Decrypted Text:", decrypted\_text)

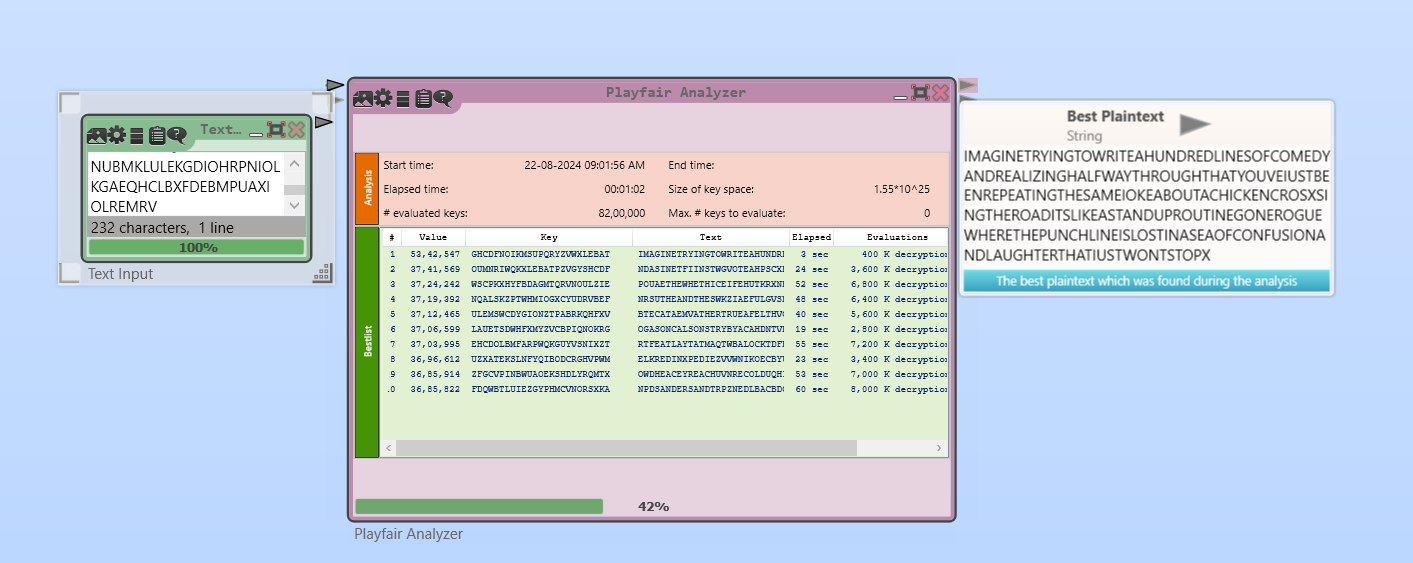
**Output:**

****

**Cryptool:**

**** ****

****

****

**Revised Approach:**

**Introduction:**

The provided code implements a modified version of the **Caesar cipher**, which is a type of substitution cipher where each letter in the plaintext is shifted by a fixed number of positions down the alphabet. This implementation enhances the basic Caesar cipher by introducing a dynamic key that varies based on the position of each character in the text.

**Drawbacks of the Simple Playfair Cipher**

1. **Vulnerability to Frequency Analysis:**

* Although Playfair encrypts digraphs instead of individual letters, patterns in the ciphertext can still be analyzed, making it vulnerable to frequency analysis and pattern recognition attacks.

1. **Limited Keyspace:**

* The Playfair cipher uses only 25 letters (excluding 'J'), resulting in a smaller keyspace compared to modern encryption methods, making it more susceptible to brute-force attacks.

1. **Predictability of Structure:**

* The regular structure of the Playfair cipher's matrices and its predictable pattern of encryption can reveal information about the message, especially if the key is not sufficiently complex.

**How This Version Improves on Simple Playfair Cipher**

1. **Dual Matrix Approach:**

* **Improvement:** By using two different 5x5 matrices derived from split parts of the key, your implementation introduces additional complexity. This alternation helps obscure patterns in the ciphertext, making it more resistant to frequency analysis and pattern detection.

1. **Random Character Addition:**

* **Improvement:** Adding a random alphabet character at the end of the ciphertext enhances obfuscation. It prevents attackers from easily identifying the encryption technique or deducing patterns, adding an extra layer of unpredictability and making cryptanalysis more challenging.

1. **Improved Ciphertext Length Handling:**

* **Improvement:** Appended Random Character: The random character added to the ciphertext not only increases security but also ensures that the length of the ciphertext isn't directly indicative of the plaintext length. This makes it harder for attackers to infer the length of the original message, adding an extra layer of obfuscation.

**Example:**

Let's walk through your modified version of the Playfair cipher using the example plaintext "ATTACK AT DAWN" and the keyword "BATTLE."

**1. Create the Matrices**

Your code first splits the keyword into two parts to create two 5x5 matrices. The keyword "BATTLE" is split as follows:

* **Key1:** "BAT"
* **Key2:** "TLE"

**Matrix 1 (using Key1 = "BAT"):**

1. Start with the letters in "BAT":

B A T

1. Then, fill in the remaining letters of the alphabet (excluding "J"):

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

**Matrix 2 (using Key2 = "TLE"):**

1. Start with the letters in "TLE":

T L E

1. Then, fill in the remaining letters of the alphabet (excluding "J"):

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

**2. Prepare the Plaintext**

The plaintext "ATTACK AT DAWN" is processed as follows:

* **Convert to Uppercase:** "ATTACK AT DAWN"
* **Replace "J" with "I":** Not applicable here since there is no "J."
* **Remove Non-Alphabetic Characters:** "ATTACKATDAWN"
* **Insert 'X' Between Duplicate Letters:** The digraphs are formed, and an 'X' is inserted between the repeated "T"s:
  + **Prepared Text:** "AT TA CK AT DA WN"

**3. Encrypt the Plaintext**

Now, let's encrypt the prepared text "ATTACK AT DAWN" (prepared as "ATTXAC KATDAWNX") using the alternating matrices:

* **First Digraph (AT):**
  + Use **Matrix 1**.
  + Find positions: A (Row 1, Column 2), T (Row 1, Column 3)
  + **Same Row Rule:** Shift right → "AT" becomes "TC".
* **Second Digraph (TA):**
  + Use **Matrix 2**.
  + Find positions: T (Row 1, Column 1), A (Row 1, Column 4)
  + **Rectangle Rule:** Swap corners → "TA" becomes "LB".
* **Third Digraph (CK):**
  + Use **Matrix 1**.
  + Find positions: C (Row 1, Column 4), K (Row 3, Column 1)
  + **Rectangle Rule:** Swap corners → "CK" becomes "BN".
* **Fourth Digraph (AT):**
  + Use **Matrix 2**.
  + Find positions: A (Row 1, Column 4), T (Row 1, Column 3)
  + **Rectangle Rule:** Swap corners → "AT" becomes "BL".
* **Fifth Digraph (DA):**
  + Use **Matrix 1**.
  + Find positions: D (Row 2, Column 2), A (Row 1, Column 2)
  + **Rectangle Rule:** Swap corners → "DA" becomes "BT".
* **Sixth Digraph (WN):**
  + Use **Matrix 2**.
  + Find positions: W (Row 5, Column 2), N (Row 3, Column 4)
  + **Rectangle Rule:** Swap corners → "WN" becomes "YK".
* **Final Ciphertext Before Random Addition:** "TCLBBNBLBTYK"
* **Random Character Addition:** A random character (e.g., "G") is appended to the ciphertext.
  + **Final Ciphertext:** " TCLBBNBLBTYKG"

**4. Decrypt the Ciphertext**

During decryption:

1. **Remove the Random Character:** Remove the last character "G."
2. **Decrypt Each Digraph:**
   * Reverse the encryption steps using the same alternating matrices to recover the original digraphs.

**Source Code:**

import random

def create\_matrices(key):

def create\_matrix(key):

matrix = []

alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ"

for char in key:

if char not in matrix and char in alphabet:

matrix.append(char)

for char in alphabet:

if char not in matrix:

matrix.append(char)

return [matrix[i:i + 5] for i in range(0, 25, 5)]

# Split the key into two parts

if len(key) % 2 == 1:

mid = len(key) // 2 + 1

else:

mid = len(key) // 2

key1 = key[:mid]

key2 = key[mid:]

matrix1 = create\_matrix(key1)

matrix2 = create\_matrix(key2)

return matrix1, matrix2

def find\_position(matrix, char):

for i in range(5):

for j in range(5):

if matrix[i][j] == char:

return i, j

return None

def encrypt(plaintext, key):

matrix1, matrix2 = create\_matrices(key)

plaintext = plaintext.upper().replace("J", "I")

# Remove all spaces and non-alphabetic characters

prepared\_text = ""

for char in plaintext:

if 'A' <= char <= 'Z':

prepared\_text += char

i = 0

while i < len(prepared\_text):

if i + 1 < len(prepared\_text) and prepared\_text[i] == prepared\_text[i + 1]:

prepared\_text = prepared\_text[:i+1] + 'X' + prepared\_text[i+1:]

i += 2

if len(prepared\_text) % 2 != 0:

prepared\_text += 'X'

ciphertext = ""

for i in range(0, len(prepared\_text), 2):

matrix = matrix1 if (i // 2) % 2 == 0 else matrix2

row1, col1 = find\_position(matrix, prepared\_text[i])

row2, col2 = find\_position(matrix, prepared\_text[i + 1])

if row1 == row2:

ciphertext += matrix[row1][(col1 + 1) % 5]

ciphertext += matrix[row2][(col2 + 1) % 5]

elif col1 == col2:

ciphertext += matrix[(row1 + 1) % 5][col1]

ciphertext += matrix[(row2 + 1) % 5][col2]

else:

ciphertext += matrix[row1][col2]

ciphertext += matrix[row2][col1]

# Generate and append a random character

alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ"

random\_char = alphabet[random.randint(0, len(alphabet) - 1)]

ciphertext += random\_char

return ciphertext

def decrypt(ciphertext, key):

matrix1, matrix2 = create\_matrices(key)

ciphertext = ciphertext[:-1] # Remove random character

prepared\_text = ""

for i in range(0, len(ciphertext), 2):

matrix = matrix1 if (i // 2) % 2 == 0 else matrix2

row1, col1 = find\_position(matrix, ciphertext[i])

row2, col2 = find\_position(matrix, ciphertext[i + 1])

if row1 == row2:

prepared\_text += matrix[row1][(col1 - 1) % 5]

prepared\_text += matrix[row2][(col2 - 1) % 5]

elif col1 == col2:

prepared\_text += matrix[(row1 - 1) % 5][col1]

prepared\_text += matrix[(row2 - 1) % 5][col2]

else:

prepared\_text += matrix[row1][col2]

prepared\_text += matrix[row2][col1]

return prepared\_text

key = "BATTLE"

plaintext = input("Enter the plain text: ")

# Modify the plaintext to remove spaces and any non-alphabetic characters

plaintext = "".join([char for char in plaintext if char.isalpha()])

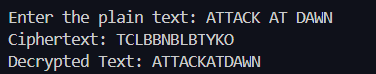
ciphertext = encrypt(plaintext, key)

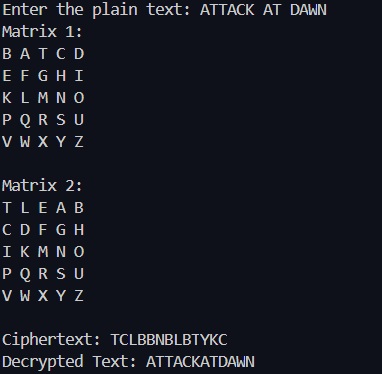
decrypted\_text = decrypt(ciphertext, key)

print("Ciphertext:", ciphertext)

print("Decrypted Text:", decrypted\_text)

**Output:**

****

****

**Cryptanalysis on PlayFair Cipher:**

1. **Pattern Matching:**

* Look for common digraphs or repeated patterns in the ciphertext. For example, in English, common digraphs include "TH," "HE," "IN," etc. Repeated patterns might suggest common digraphs in the plaintext.

1. **Known Plaintext Attack:**

* If you have a sample of plaintext and its corresponding ciphertext, you can use this to deduce parts of the key matrix. For example, if you know the plaintext "HELLO" and see its ciphertext, you can reverse-engineer the positions of the letters in the matrix.

1. **Brute Force:**

* Given the limited number of possible 5x5 matrices (around 150,000), a brute force attack might be feasible. Generate all possible matrices, decrypt the ciphertext with each matrix, and check if the result makes sense.

1. **Matrix Manipulation:**

* Use matrix algebra to analyze the relationships between ciphertext and plaintext pairs. For example, if you can guess some plaintext-ciphertext pairs, use them to construct and solve linear equations to find the key matrix.

**Conclusion:**

This modified version of the Playfair cipher significantly enhances the security and complexity of the traditional encryption method. By introducing dual matrices, it makes the encryption process more resistant to pattern recognition and frequency analysis, two common weaknesses in the original Playfair cipher. The addition of a random character at the end of the ciphertext further obscures the encryption technique, adding layer of unpredictability that makes it more challenging for attackers to deduce the method used. Moreover, the careful handling of duplicate letters and filler characters ensures that the encryption and decryption processes are both consistent and accurate, preserving the integrity of the original message.

In summary, this version addresses several critical drawbacks of the traditional Playfair cipher, offering a more secure and sophisticated approach to encryption that is better suited to resist modern cryptographic attacks.

**References:**

1. [An-Improved-3D-Playfair-Cipher-Key-Matrix-with-Dual-Cipher-Block-Chaining-Method.pdf (researchgate.net)](https://www.researchgate.net/profile/Ritchell-Villafuerte/publication/338621994_An_Improved_3D_Playfair_Cipher_Key_Matrix_with_Dual_Cipher_Block_Chaining_Method/links/5f60d7b4299bf1d43c0594f8/An-Improved-3D-Playfair-Cipher-Key-Matrix-with-Dual-Cipher-Block-Chaining-Method.pdf): (Ritchell S. Villafuerte, Ariel M. Sison, Ruji P. Medina)
2. [IEEE Xplore Full-Text PDF:](https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6612193) (Amandeep Kaur, Harsh Kumar Verma, Ravindra Kumar Singh From Dept. of Computer Science & Engineering National Institute of Technology, Jalandhar, Punjab)