



KING FAISAL UNIVERSITY

College of Computer Sciences and Information Technology (CCSIT)

| | |
|-----------------|--------------------------------------|
| Name1: | Esraa Ali Albahrani-221425419 |
| Name2: | Thana Essa Alradhi-219023255 |
| Section: | 64 |

| | |
|--|----|
| INTRODUCTION | 3 |
| Analysis of frequency of letters in a text paragraph..... | 4 |
| Results of running the code | 5 |
| PLAYFAIR CIPHER..... | 6 |
| How it works | 6 |
| Rules for Encryption..... | 6 |
| Limitations of the Cipher. | 7 |
| Implementing the code for Playfair Cipher..... | 9 |
| How the code works. | 10 |
| Encryption | 11 |
| Decryption | 11 |
| How the Playfair cipher compare to other classical ciphers | 11 |
| CONCLUSION | 13 |
| REFERENCES | 14 |

INTRODUCTION

This project delves into an in-depth analysis of the Playfair Cipher, a classical method of encryption, and juxtaposes it with other historical encryption techniques. The Playfair Cipher, named after its inventor Charles Wheatstone, is a fascinating study in itself, offering a glimpse into the ingenuity of early cryptographic systems. The cipher's unique mechanism of encrypting pairs of letters, rather than single ones, sets it apart from many of its contemporaries. By studying this and other such classical ciphers, we gain a profound understanding of the evolution of cryptographic systems. This understanding is crucial as it provides a foundation upon which modern cryptography is built.

Understanding the history of encryption is not just about appreciating the complexity of modern cryptographic systems, but also about recognizing the progress of technology. The journey from simple substitution ciphers to complex algorithms involving computational number theory is a testament to human innovation. The primary objective of demonstrating and analyzing the Playfair Cipher is to highlight this technological advancement. In addition to this, the project also undertakes an analysis of letter frequencies. This task underscores the importance of data analysis skills in cryptography. By studying the frequency of letters in encrypted messages, one can often discern patterns and potentially crack the cipher. This aspect of the project serves as a reminder that while technology has advanced, some fundamental principles remain the same. Thus, through the study of the Playfair Cipher and letter frequency analysis, this project aims to provide a comprehensive overview of both the history and the current state of cryptography.

Analysis of frequency of letters in a text paragraph

Program to count the frequency of letters in each Text paragraph, and map top highest 5 characters against E –T- A- I – O.

```
import tkinter as tk
from tkinter import messagebox
from collections import Counter
import string
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

def analyze_text():
    text = text_input.get("1.0", tk.END).upper()
    # Remove letters only
    letters = [ch for ch in text if ch in string.ascii_uppercase]
    frequency = Counter(letters)
    total_letters = sum(frequency.values())

    # Sort the frequency dictionary by frequency count in descending order
    sorted_freq = sorted(frequency.items(), key=lambda x: x[1], reverse=True)

    # Prepare data for display
    top_five = sorted_freq[:5]
    results = []
    labels = ['E', 'T', 'A', 'I', 'O']
    data_labels = []
    data_values = []

    for i, (letter, count) in enumerate(top_five):
        if i < len(labels):
            percentage = (count / total_letters) * 100 if total_letters > 0 else 0
            results.append(f'{labels[i]} ({letter}): {percentage:2f}%')
            data_labels.append(f'{labels[i]} ({letter})')
            data_values.append(percentage)

    # Update results display
    results_output.config(state=tk.NORMAL)
    results_output.delete("1.0", tk.END)
    results_output.insert("1.0", "\n".join(results))
    results_output.config(state=tk.DISABLED)

    # Display the bar chart
    show_bar_chart(data_labels, data_values)

def show_bar_chart(data_labels, data_values):
    def show_bar_chart(labels, values):
        fig, ax = plt.subplots()
        ax.bar(labels, values, color='blue')
        ax.set_xlabel('Letters')
        ax.set_ylabel('Percentage')
        ax.set_title('Top 5 Letter Frequencies')

    # Integrate matplotlib into tkinter
    canvas = FigureCanvasTkAgg(fig, master=root) # A tk.DrawingArea.
    canvas_widget = canvas.get_tk_widget()
    canvas_widget.pack(side=tk.TOP, fill=tk.BOTH, expand=1)
    canvas.draw()

# Create the main window
root = tk.Tk()
root.title("Letter Frequency Analyzer")

# Create a text input area
text_input = tk.Text(root, height=10, width=50)
text_input.pack(pady=10)

# Create a button to trigger analysis
analyze_button = tk.Button(root, text="Analyze Frequencies", command=analyze_text)
analyze_button.pack(pady=10)

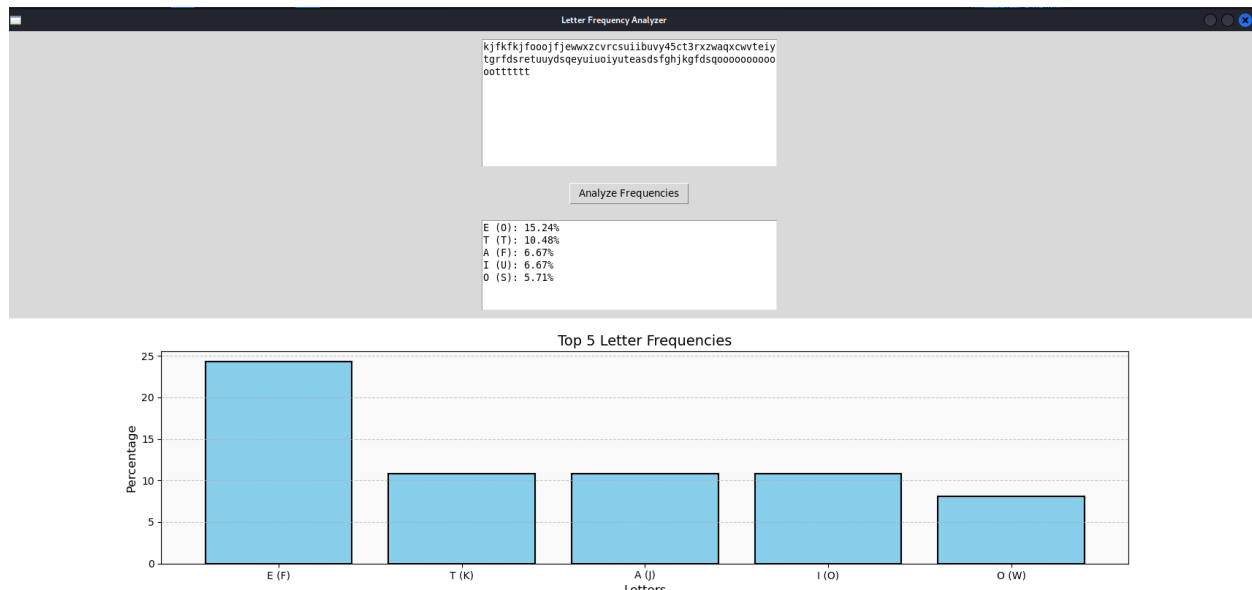
# Create an output area for the results
results_output = tk.Text(root, height=7, width=50, state=tk.DISABLED)
results_output.pack(pady=10)

# Start the GUI event loop
root.mainloop()
```

How the code works.

1. The user inputs text into the text_input area.
2. When the user clicks the “Analyze Frequencies” button, the analyze_text() function is called.
3. The function:
 - Converts the input text to uppercase.
 - Counts the occurrences of each letter.
 - Sorts the letter frequencies in descending order.
 - Displays the top five most frequent letters along with their percentages.
 - Shows a bar chart visualizing the letter frequencies.

Results of running the code



When you run the code type a paragraph, clicking the analyze button displays the results of the top letters used in a graph.

PLAYFAIR CIPHER

Historical Significance: The Playfair cipher was the first practical digraph substitution cipher¹². It was invented in 1854 by Charles Wheatstone but was named after Lord Playfair who promoted its use.

How it works:

1. Key Square (5×5):

- The key square is a 5×5 grid of unique alphabets that serves as the key for encrypting the plaintext.
- The table typically excludes one letter, often 'J', due to its capacity to hold only 25 letters. If the plaintext includes 'J', it is substituted with 'I'.
- In the key square, the first letters are the distinct letters from the key, arranged in the sequence they appear. This is then followed by the rest of the letters of the alphabet in their usual order.

2. Encryption Process:

- The plaintext is divided into two-letter groups, known as digraphs. If the total number of letters is odd, a 'Z' is appended to the final letter.
- If a pair cannot be made with the same letter, we break the letter into a single letter and add a bogus letter to the previous one.
- If a letter stands alone during pairing, we add an extra bogus letter to it.

Rules for Encryption:

- If both letters fall within the same column, select the letter directly beneath each one, cycling back to the top if you reach the bottom.
- If both letters are in the same row, choose the letter immediately to the right of each, returning to the far left if you're at the far right.
- If neither rule applies, create a rectangle using the two letters and take the letters from the horizontally opposite corners of the rectangle.

3. Example:

Encrypting the word "instruments":

- After splitting: 'in' 'st' 'ru' 'me' 'nt' 'sz'
- Applying the rules:
- "me" becomes "cl"
- "st" becomes "tl"
- "nt" becomes "rq"
- Encrypted text: "gatlmzclrqtx"

Limitations of the Cipher.

The **Playfair cipher** has several limitations, which are important to consider:

- **Key Length and Reusability:**
 - The key square (5×5 grid) used in the Playfair cipher must be carefully chosen. If an insecure or easily guessable key is used, the entire encryption becomes vulnerable.
 - Additionally, the key cannot be reused for different messages. If the same key is used repeatedly, it weakens the security of subsequent messages.
- **Limited Alphabet and Key Space:**
 - The Playfair cipher operates on a 5×5 grid, which means it can only handle 25 unique alphabets (excluding J, which is often replaced by I).
 - This limitation reduces the overall strength of the encryption compared to ciphers that use the full 26-letter alphabet.
- **Frequency Analysis:**
 - Although the Playfair cipher improves upon simple substitution ciphers, it is still susceptible to frequency analysis.
 - Digraphs (pairs of letters) tend to occur with varying frequencies in natural language. Attackers can analyze the frequency distribution of digraphs in the ciphertext to make educated guesses about the plaintext.
- **Known Plaintext Attacks:**
 - If an attacker knows part of the plaintext (known plaintext), they can exploit this information to deduce parts of the key or even the entire key.
 - For example, if the attacker knows that a certain digraph corresponds to a common word like "the," they can infer the corresponding key square entries.

- **Lack of Diffusion:**
 - Diffusion refers to how changes in the plaintext affect the entire ciphertext. In the Playfair cipher, a change in one letter only affects the adjacent letter in the ciphertext.
 - This lack of diffusion makes it easier for attackers to analyze patterns and relationships between adjacent digraphs.
- **Complexity for Manual Encryption/Decryption:**
 - While the Playfair cipher is straightforward to implement in software, manual encryption and decryption can be cumbersome.
 - The rules for pairing letters and handling edge cases (e.g., repeated letters, odd-length plaintext) require careful attention.
- **Security Depends on Key Management:**
 - The security of the Playfair cipher heavily relies on the secrecy and randomness of the key.
 - If the key is compromised or shared improperly, the entire system becomes vulnerable.

Regardless of the limitations, the Playfair cipher remains an interesting historical encryption technique. It is essential to understand its strengths and weaknesses when evaluating its suitability for specific use cases.

Implementing the code for Playfair Cipher

```
import tkinter as tk
from tkinter import simpledialog

def create_key_grid(keyword):
    alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    key_grid = ''
    seen = set()

    # Add keyword letters to the grid, skipping duplicates and 'J'
    for char in keyword.upper():
        if char not in seen and char in alphabet:
            seen.add(char)
            key_grid += char

    # Add remaining letters of the alphabet
    for char in alphabet:
        if char not in seen:
            key_grid += char

    return key_grid

def find_position(char, key_grid):
    index = key_grid.index(char)
    return index // 5, index % 5

def playfair_encrypt(plaintext, key_grid):
    plaintext = plaintext.upper().replace('J', 'I')
    ciphertext = ''
    i = 0
    while i < len(plaintext):
        a = plaintext[i]
        b = ''
        if i+1 < len(plaintext):
            b = plaintext[i+1]
        if a == b or b == '':
            ciphertext += a * 2
            i += 1
        else:
            row1, col1 = find_position(a, key_grid)
            row2, col2 = find_position(b, key_grid)
            if row1 == row2:
                ciphertext += key_grid[(row1*5 + (col1+1)%5)]
                ciphertext += key_grid[(row2*5 + (col2+1)%5)]
            elif col1 == col2:
                ciphertext += key_grid[((row1+1)%5)*5 + col1]
                ciphertext += key_grid[((row2+1)%5)*5 + col2]
            else:
                ciphertext += key_grid[(row1*5 + col2)]
                ciphertext += key_grid[(row2*5 + col1)]
            i += 2
        ciphertext += ' '
    return ciphertext

def playfair_decrypt(ciphertext, key_grid):
    plaintext = ''
    i = 0
    while i < len(ciphertext):
        a = ciphertext[i]
        b = ciphertext[i+1]
        i += 2

        row1, col1 = find_position(a, key_grid)
        row2, col2 = find_position(b, key_grid)

        if row1 == row2:
            plaintext += key_grid[(row1*5 + (col1-1)%5)]
            plaintext += key_grid[(row2*5 + (col2-1)%5)]
        elif col1 == col2:
            plaintext += key_grid[((row1-1)%5)*5 + col1]
            plaintext += key_grid[((row2-1)%5)*5 + col2]
        else:
            plaintext += key_grid[(row1*5 + col2)]
            plaintext += key_grid[(row2*5 + col1)]
    return plaintext
```

```
root@kali: /home/kali/Documents x root@kali: /home/kali/Documents x root@kali: /home/kali/Documents x
GNU nano 7.2 playfair.py
if a == b or b == '':
    b = 'X'
    i += 1
else:
    i += 2

row1, col1 = find_position(a, key_grid)
row2, col2 = find_position(b, key_grid)

if row1 == row2:
    ciphertext += key_grid[(row1*5 + (col1+1)%5)]
    ciphertext += key_grid[(row2*5 + (col2+1)%5)]
elif col1 == col2:
    ciphertext += key_grid[((row1+1)%5)*5 + col1]
    ciphertext += key_grid[((row2+1)%5)*5 + col2]
else:
    ciphertext += key_grid[(row1*5 + col2)]
    ciphertext += key_grid[(row2*5 + col1)]

return ciphertext

def playfair_decrypt(ciphertext, key_grid):
    plaintext = ''
    i = 0
    while i < len(ciphertext):
        a = ciphertext[i]
        b = ciphertext[i+1]
        i += 2

        row1, col1 = find_position(a, key_grid)
        row2, col2 = find_position(b, key_grid)

        if row1 == row2:
            plaintext += key_grid[(row1*5 + (col1-1)%5)]
            plaintext += key_grid[(row2*5 + (col2-1)%5)]
        elif col1 == col2:
            plaintext += key_grid[((row1-1)%5)*5 + col1]
            plaintext += key_grid[((row2-1)%5)*5 + col2]
        else:
            plaintext += key_grid[(row1*5 + col2)]
            plaintext += key_grid[(row2*5 + col1)]
    return plaintext

Help Write Out Where Is Cut Undo Location M-U Undo Set Mark M-] To Bracket M-; Previous P- Back P- Forward Prev Word
Exit Read File Replace Paste Justify Go To Line Redo Copy Where Was Next Forward Next Word
```

```

root@kali: /home/kali/Documents x root@kali: /home/kali/Documents x root@kali: /home/kali/Documents x
GNU nano 7.2 playfair.py
if a == b or b == '':
    b = 'X'
    i += 1
else:
    i += 2

row1, col1 = find_position(a, key_grid)
row2, col2 = find_position(b, key_grid)

if row1 == row2:
    ciphertext += key_grid[row1*5 + (col1+1)%5]
    ciphertext += key_grid[row2*5 + (col2+1)%5]
elif col1 == col2:
    ciphertext += key_grid[((row1+1)%5)*5 + col1]
    ciphertext += key_grid[((row2+1)%5)*5 + col2]
else:
    ciphertext += key_grid[row1*5 + col2]
    ciphertext += key_grid[row2*5 + col1]

return ciphertext

def playfair_decrypt(ciphertext, key_grid):
    plaintext = ''
    i = 0
    while i < len(ciphertext):
        a = ciphertext[i]
        b = ciphertext[i+1]
        i += 2

        row1, col1 = find_position(a, key_grid)
        row2, col2 = find_position(b, key_grid)

        if row1 == row2:
            plaintext += key_grid[row1*5 + (col1-1)%5]
            plaintext += key_grid[row2*5 + (col2-1)%5]
        elif col1 == col2:
            plaintext += key_grid[((row1-1)%5)*5 + col1]
            plaintext += key_grid[((row2-1)%5)*5 + col2]
        else:
            plaintext += key_grid[row1*5 + col2]
            plaintext += key_grid[row2*5 + col1]

    return plaintext

root = tk.Tk()
root.title("Playfair Cipher GUI")

def encrypt_action():
    keyword = keyword_entry.get()
    plaintext = plaintext_entry.get()
    key_grid = create_key_grid(keyword)
    encrypted_text = playfair_encrypt(plaintext, key_grid)
    result_label.config(text=f'Encrypted: {encrypted_text}')

def decrypt_action():
    keyword = keyword_entry.get()
    ciphertext = ciphertext_entry.get()
    key_grid = create_key_grid(keyword)
    decrypted_text = playfair_decrypt(ciphertext, key_grid)
    result_label.config(text=f'Decrypted: {decrypted_text}')

tk.Label(root, text="Keyword:").grid(row=0, column=0)
keyword_entry = tk.Entry(root)
keyword_entry.grid(row=0, column=1)

tk.Label(root, text="Plaintext / Ciphertext:").grid(row=1, column=0)
plaintext_entry = tk.Entry(root)
plaintext_entry.grid(row=1, column=1)

encrypt_button = tk.Button(root, text="Encrypt", command=encrypt_action)
encrypt_button.grid(row=2, column=0)

decrypt_button = tk.Button(root, text="Decrypt", command=decrypt_action)
decrypt_button.grid(row=2, column=1)

result_label = tk.Label(root, text="Result will appear here...")
result_label.grid(row=3, column=0, columnspan=2)

root.mainloop()

if __name__ == "__main__":
    main()

```

How the code works.

- `create_key_grid(keyword)`: This function creates a 5x5 grid of letters based on a keyword. The keyword letters are added first (ignoring duplicates and 'J'), followed by the remaining letters of the alphabet.
- `find_position(char, key_grid)`: This function finds the position of a character in the key grid and returns its row and column indices.
- `playfair_encrypt(plaintext, key_grid)`: This function encrypts a plaintext message using the Playfair cipher. It processes the plaintext two characters at a time (adding an 'X' if necessary), finds their positions in the key grid, and applies the Playfair encryption rules to generate the ciphertext.
- `playfair_decrypt(ciphertext, key_grid)`: This function decrypts a ciphertext message using the Playfair cipher. It processes the ciphertext two characters at a time, finds their positions in the key grid, and applies the Playfair decryption rules to generate the plaintext.

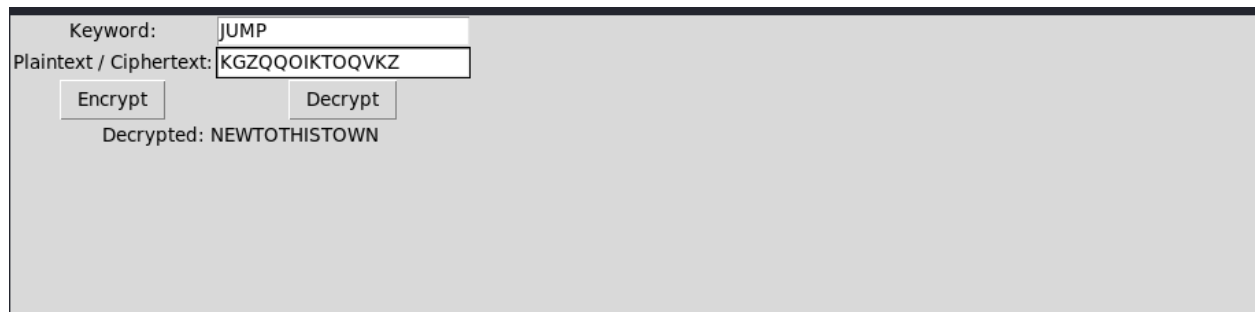
- `main()`: This function creates a Tkinter GUI for the user to input a keyword and a plaintext or ciphertext message. The user can then click the “Encrypt” or “Decrypt” button to perform the corresponding operation. The result is displayed in the GUI.

Encryption



The screenshot shows a Tkinter GUI for encryption. It has a label 'Keyword:' with a text entry field containing 'JUMP'. Below it is a label 'Plaintext / Ciphertext:' with a text entry field containing 'NEWTOTHISTOWN'. There are two buttons: 'Encrypt' and 'Decrypt'. Below the buttons, the text 'Encrypted: KGZQQOIKTOQVKZ' is displayed.

Decryption



The screenshot shows the same Tkinter GUI but for decryption. The 'Keyword:' field still contains 'JUMP'. The 'Plaintext / Ciphertext:' field now contains 'KGZQQOIKTOQVKZ'. The 'Decrypt' button is highlighted. Below the buttons, the text 'Decrypted: NEWTOTHISTOWN' is displayed.

How the Playfair cipher compare to other classical ciphers

- **Substitution Ciphers:**
 - The Playfair cipher is an improvement over simple substitution ciphers (like the Caesar cipher) because it operates on digraphs (pairs of letters) rather than individual letters.
 - In a substitution cipher, each letter is replaced by another letter according to a fixed rule (e.g., shifting by a fixed amount).
 - The Playfair cipher provides better security due to its digraph-based approach, but it still suffers from frequency analysis vulnerabilities.

- **Vigenère Cipher:**
 - The Vigenère cipher is a polyalphabetic substitution cipher that uses a keyword to determine the shift for each letter.
 - Unlike the Playfair cipher, which operates on digraphs, the Vigenère cipher encrypts individual letters.
 - The Vigenère cipher is more resistant to frequency analysis because it introduces variability based on the keyword.
- **Transposition Ciphers:**
 - Transposition ciphers (such as the Rail Fence cipher) rearrange the order of letters in the plaintext without altering the letters themselves.
 - Unlike substitution ciphers, transposition ciphers do not replace letters with other letters.
 - The Playfair cipher focuses on letter replacement, while transposition ciphers focus on reordering.
- **Strengths of the Playfair Cipher:**
 - Digraph-based encryption: The Playfair cipher provides better security than simple substitution ciphers by working with pairs of letters.
 - Speed: It was historically used for tactical purposes due to its speed and simplicity.
 - Key management: The key square serves as the secret key, making it easy to manage.
- **Weaknesses of the Playfair Cipher:**
 - Limited alphabet: The 5×5 key square restricts the alphabet to 25 letters (excluding J/I). This reduces the overall strength compared to ciphers using the full 26-letter alphabet.
 - Vulnerability to known plaintext attacks: If an attacker knows part of the plaintext, they can deduce parts of the key.
 - Lack of diffusion: Changes in one letter affect only adjacent letters in the ciphertext.
- **Overall Comparison:**
 - The Playfair cipher strikes a balance between simplicity and security. While it is an improvement over basic substitution ciphers, it is not as robust as modern encryption methods.
 - For stronger security, modern ciphers like the Advanced Encryption Standard (AES) or RSA are preferred.

CONCLUSION

Encryption Method: The Playfair cipher differs from traditional ciphers by encrypting pairs of alphabets (digraphs) instead of a single alphabet. This complexity makes it much more difficult to decipher, as the frequency analysis typically employed for simple substitution ciphers is ineffective against it.

The Playfair cipher's role in warfare: Employed tactically by British units in the Second Boer War and World War I, as well as by Australian forces in World War II, the Playfair cipher was favored for its speed and the absence of a need for specialized equipment. This underscores the significant impact that cryptography has had on historical events.

Advancement over Other Methods: The Playfair cipher shows a great advancement over other encryption methods, for example the Caesar Cipher.

REFERENCES

1. Sobel, Martin. Codes, Ciphers and Secret Writing. Dover Publications, 1984.
2. D'Agapeyeff, Alexander. Codes and Ciphers: A History of Cryptography. Oxford University Press, 1939.
3. Banoth, Rajkumar. Classical and Modern Cryptography for Beginners. Independently Published, 2021.
4. Diepenbroek, Martine. The Spartan Scytale and Developments in Ancient and Modern Cryptography. Independently Published, 2022.
5. Bhat, Abhay. " Playfair Cipher with Example." Geeks for Geeks, 06 Mar. 2024, <https://www.geeksforgeeks.org/playfair-cipher-with-examples/>.