BHARATIYA VIDYA BHAVAN'S

SARDAR PATEL INSTITUTE OF TECHNOLOGY

(Empowered Autonomous Institute Affiliated to University of Mumbai)

[Knowledge is Nectar]

Department of Computer Science and Engineering

UID	2024801016
Name	Rayyan Ahmed Siddiqui
Class and Batch	TE Computer Science and Engineering - Batch B3
Date	2nd September 2025
Lab #	2
Aim	***Experiment on Symmetric Ciphers***
Objective	***To implement all substitution encryption techniques namely Caesar Ciphers, Monoalphabetic Cipers, Playfair Cipher, Hill Cipher and Polyalphabetic Ciphers. Then, perform ethical hacking on all substitution encryption techniques.***
Theory	In Network Security Model, encryption and decryption play important role of sending message so that other cannot see them. When encryption and decryption is performed by the same key it is called symmetric cryptosystem. There is a class of symmetric encryption cryptosystem where each letter of plain text is substituted to another letter called as substitution encryption techniques.
Procedure	Caesar Cipher is one of the simplest and oldest methods of encrypting messages, named after Julius Caesar, who reportedly used it to protect his military communications. This technique involves shifting the letters of the alphabet by a fixed number of places. For example, with a shift of three, the letter 'A' becomes 'D', 'B' becomes 'E', and so on. Monoalphabetic substitution is a cipher in which each occurrence of a plaintext symbol is replaced by a corresponding ciphertext symbol to generate cipher text. The key for such a cipher is a table of the correspondence or a function from which the correspondence is computed. In playfair cipher unlike traditional cipher we encrypt a pair of alphabets(digraphs) instead of a single alphabet. Hill cipher is a polygraphic substitution cipher based on linear algebra. Each letter is represented by a number modulo 26. To encrypt a message, each block of n letters (considered as an n-component vector) is multiplied by an invertible n × n matrix, against modulus 26. To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption. The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible n × n matrices (modulo 26). Polyalphabetic Cipher is a cipher where each letter in the plaintext can be encrypted to multiple possible letters in the ciphertext, depending on its position and a more

complex algorithm. In this article, we will see the differences between Monoalphabetic Cipher and Polyalphabetic Cipher

Part 1

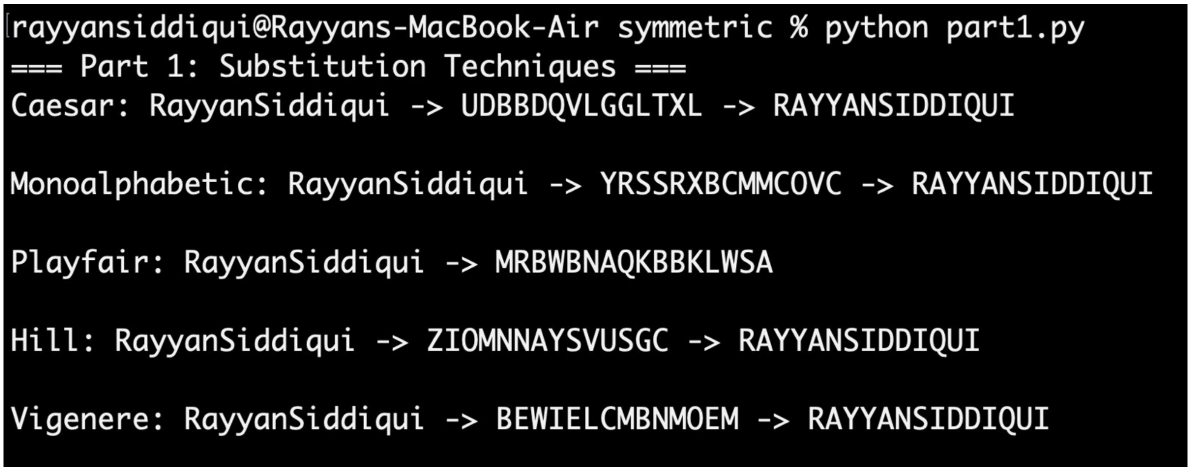
Code

```
import random
import string
import numpy as np


# ------------------------
# Caesar Cipher
# ------------------------
def caesar_encrypt(text, key):
    result = ""
    for ch in text.upper():
        if ch.isalpha():
            result += chr((ord(ch) - 65 + key) % 26 + 65)
        else:
            result += ch
    return result


def caesar_decrypt(cipher, key):
    return caesar_encrypt(cipher, -key)


# ------------------------
# Monoalphabetic Cipher
# ------------------------
def generate_mono_key():
    letters = list(string.ascii_uppercase)
    shuffled = letters[:]
    random.shuffle(shuffled)
    return dict(zip(letters, shuffled)), dict(zip(shuffled, letters))


def mono_encrypt(text, keymap):
    return "".join(keymap.get(ch, ch) for ch in text.upper())

def mono_decrypt(cipher, revmap):
    return "".join(revmap.get(ch, ch) for ch in cipher.upper())


# ------------------------
# Playfair Cipher
# ------------------------
def generate_playfair_matrix(key):
    key = "".join(dict.fromkeys(key.upper().replace("J", "I") +
string.ascii_uppercase.replace("J","")))
    matrix = [key[i:i+5] for i in range(0, 25, 5)]
    return matrix


def playfair_encrypt_pair(a, b, matrix):
    if a == b: b = "X"
    for r in range(5):
        for c in range(5):
            if matrix[r][c] == a:
                ra, ca = r, c
            if matrix[r][c] == b:
                rb, cb = r, c
    if ra == rb:
        return matrix[ra][(ca+1)%5] + matrix[rb][(cb+1)%5]
    elif ca == cb:
        return matrix[(ra+1)%5][ca] + matrix[(rb+1)%5][cb]
    else:
        return matrix[ra][cb] + matrix[rb][ca]


def playfair_encrypt(text, key):
    matrix = generate_playfair_matrix(key)
    text = text.upper().replace("J","I")
    pairs = []
    i = 0
    while i < len(text):
        a = text[i]
        b = text[i+1] if i+1 < len(text) else "X"
        if a == b:
            pairs.append((a,"X"))
            i += 1
        else:
            pairs.append((a,b))
            i += 2
    return "".join(playfair_encrypt_pair(a,b,matrix) for a,b in pairs)


# ------------------------
# Hill Cipher
# ------------------------
def hill_encrypt(text, key_matrix):
    text = text.upper().replace(" ", "")
    while len(text) % len(key_matrix) != 0:
        text += "X"
    numbers = [ord(c)-65 for c in text]
    result = ""
    n = len(key_matrix)
    for i in range(0, len(numbers), n):
        block = np.array(numbers[i:i+n])
        enc = np.dot(key_matrix, block) % 26
        result += "".join(chr(num+65) for num in enc)
    return result


def mod_inverse_matrix(matrix, mod=26):
    det = int(round(np.linalg.det(matrix)))
    det_inv = pow(det % mod, -1, mod)
    matrix_mod = np.round(det * np.linalg.inv(matrix)).astype(int) % mod
    return (det_inv * matrix_mod) % mod


def hill_decrypt(cipher, key_matrix):
    inv_matrix = mod_inverse_matrix(key_matrix, 26)
    numbers = [ord(c)-65 for c in cipher]
    result = ""
    n = len(key_matrix)
    for i in range(0, len(numbers), n):
        block = np.array(numbers[i:i+n])
        dec = np.dot(inv_matrix, block) % 26
        result += "".join(chr(num+65) for num in dec)
    return result


# ------------------------
# Vigenere (Polyalphabetic)
# ------------------------
def vigenere_encrypt(text, key):
    text, key = text.upper(), key.upper()
    result = ""
    for i, ch in enumerate(text):
        if ch.isalpha():
            shift = ord(key[i % len(key)]) - 65
            result += chr((ord(ch)-65+shift)%26 + 65)
        else:
            result += ch
    return result
```

```
def vigenere_decrypt(cipher, key):
    text, key = cipher.upper(), key.upper()
    result = ""
    for i, ch in enumerate(text):
        if ch.isalpha():
            shift = ord(key[i % len(key)]) - 65
            result += chr((ord(ch)-65-shift)%26 + 65)
        else:
            result += ch
    return result




# ------------------------
# DEMO
# ------------------------
if __name__ == "__main__":
    print("=== Part 1: Substitution Techniques ===")


    # Caesar
    msg = "RayyanSiddiqui"
    c_key = 3
    caes_enc = caesar_encrypt(msg, c_key)
    print(f"Caesar: {msg} -> {caes_enc} -> {caesar_decrypt(caes_enc, c_key)}\n")


    # Monoalphabetic
    keymap, revmap = generate_mono_key()
    mono_enc = mono_encrypt(msg, keymap)
    print(f"Monoalphabetic: {msg} -> {mono_enc} -> {mono_decrypt(mono_enc, revmap)}\n")


    # Playfair
    pf_key = "MONARCHY"
    pf_enc = playfair_encrypt(msg, pf_key)
    print(f"Playfair: {msg} -> {pf_enc}\n")


    # Hill
    hill_key = np.array([[3,3],[2,5]])  # example invertible matrix
    hill_enc = hill_encrypt(msg, hill_key)
    print(f"Hill: {msg} -> {hill_enc} -> {hill_decrypt(hill_enc, hill_key)}\n")
```

```
# Vigenere
v_key = "KEY"
vig_enc = vigenere_encrypt(msg, v_key)
print(f"Vigenere: {msg} -> {vig_enc} -> {vigenere_decrypt(vig_enc, v_key)}\n")
```

Output



```
[rayyansiddiqui@Rayyans-MacBook-Air symmetric % python part1.py
=== Part 1: Substitution Techniques ===
Caesar: RayyanSiddiqui -> UDBBDQVLGGLTXL -> RAYYANSIDDIQUI

Monoalphabetic: RayyanSiddiqui -> YRSSRXBCMMCOVC -> RAYYANSIDDIQUI

Playfair: RayyanSiddiqui -> MRBWBNAQKBBKLWSA

Hill: RayyanSiddiqui -> ZIOMNNAYSVUSGC -> RAYYANSIDDIQUI

Vigenere: RayyanSiddiqui -> BEWIELCMBNMOEM -> RAYYANSIDDIQUI
```

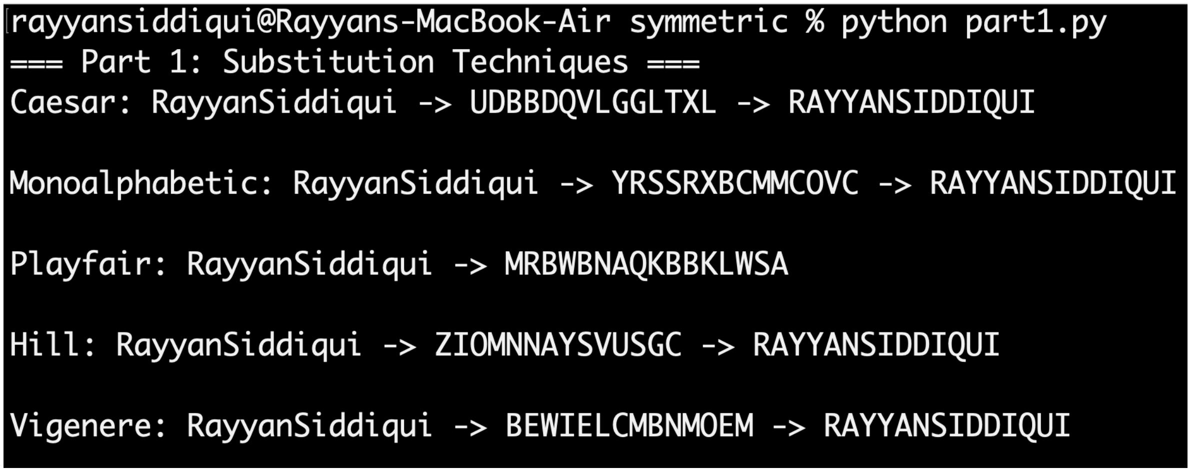
Part 2

Code

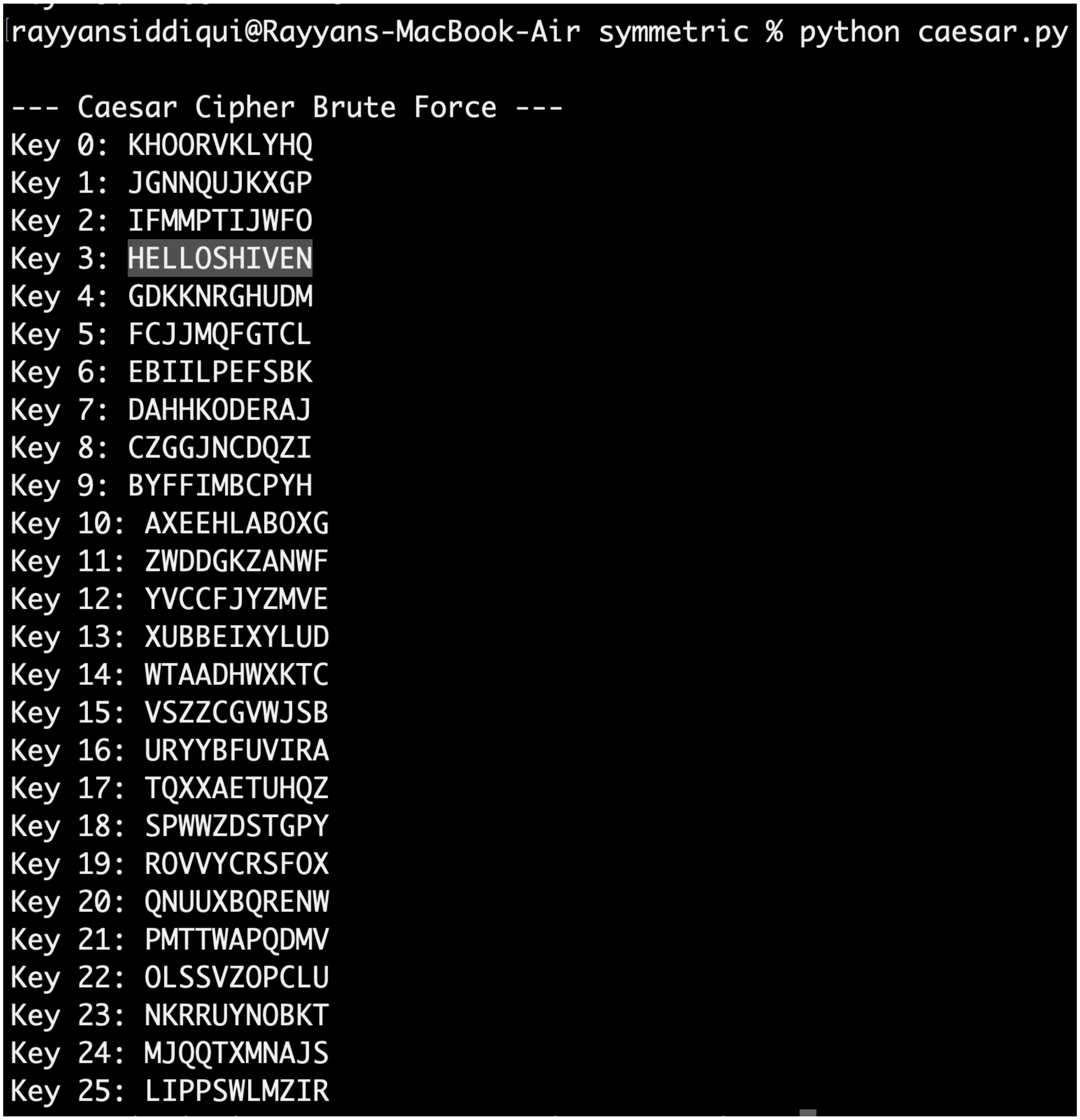
```
//caeser
def caesar_bruteforce(ciphertext):
    print("\n--- Caesar Cipher Brute Force ---")
    for key in range(26):
        translated = ""
        for symbol in ciphertext:
            if symbol.isalpha():
                base = ord('A') if symbol.isupper() else ord('a')
                translated += chr((ord(symbol) - base - key) % 26 + base)
            else:
                translated += symbol
        print(f"Key {key}: {translated}")

cipher = "KHOORVKLYHQ"   #MESSAGE ENCRYPTED:"HELLOSHIVEN"
caesar_bruteforce(cipher)


//playfair
def playfair_frequency_attack(ciphertext):
    text = ''.join([c.upper() for c in ciphertext if c.isalpha()])
    digrams = [text[i:i+2] for i in range(0, len(text), 2)]
    from collections import Counter
    freq = Counter(digrams)
    print("\n--- Playfair Frequency Analysis ---")
    print("Top ciphertext digrams:")
    for digram, count in freq.most_common(10):
        print(f"{digram}: {count}")

#MESSAGE ENCRYPTED: "HELLOSHIVEN"
cipher = "CFSUBPMPBFXGM"
playfair_frequency_attack(cipher)


//vignere
def vigenere_decrypt(ciphertext, key):
    result = ""
    key = key.upper()
    ki = 0
    for ch in ciphertext:
        if ch.isalpha():
            base = ord('A') if ch.isupper() else ord('a')
            shift = ord(key[ki % len(key)]) - ord('A')
            result += chr((ord(ch) - base - shift) % 26 + base)
            ki += 1
        else:
            result += ch
    return result

def vigenere_dictionary_attack(ciphertext, dictionary=["KEY", "SECRET", "HELLO"]):
    print("\n--- Vigenère Dictionary Attack ---")
    for word in dictionary:
        guess = vigenere_decrypt(ciphertext, word)
        print(f"Key '{word}': {guess}")

cipher = "RIJVSQRMTOR"   #MESSAGE ENCRYPTED: "HELLOSHIVEN"
dictionary = ["KEY", "HELLO", "SECRET"]
vigenere_dictionary_attack(cipher, dictionary)


//monoalphabetic
from collections import Counter

def mono_freq_attack(ciphertext):
    text = ''.join([c.upper() for c in ciphertext if c.isalpha()])
    freq = Counter(text)
    print("\n--- Monoalphabetic Cipher Frequency ---")
    print("Cipher frequencies:", freq.most_common())
```

```
# Example ciphertext
cipher = "ZJBBINZQFJM"   #MESSAGE ENCRYPTED: "HELLOSHIVEN"
mono_freq_attack(cipher)


//hillcipher
import numpy as np

def to_nums(text):
    return [ord(c) - 65 for c in text.upper() if c.isalpha()]

def to_text(nums):
    return ''.join(chr(n + 65) for n in nums)

def hill_encrypt(plaintext, K, n):
    nums = to_nums(plaintext)
    nums = nums[:(len(nums)//n)*n]  # make divisible by n
    P = np.array(nums).reshape(-1, n).T
    C = (K.dot(P)) % 26
    return to_text(C.T.flatten())

def mod_inverse_matrix(matrix, mod=26):
    """Find modular inverse of matrix (integer method)."""
    det = int(round(np.linalg.det(matrix))) % mod
    det_inv = pow(det, -1, mod)  # modular inverse of determinant

    # adjugate (cofactor transpose)
    n = matrix.shape[0]
    adj = np.zeros((n, n), dtype=int)
    for r in range(n):
        for c in range(n):
            minor = np.delete(np.delete(matrix, r, axis=0), c, axis=1)
            cofactor = int(round(np.linalg.det(minor)))
            adj[c, r] = ((-1) ** (r + c)) * cofactor
    return (det_inv * adj) % mod

def hill_attack(plaintext, ciphertext, n):
    """Recover Hill cipher key matrix from known plaintext-ciphertext pairs."""
    p_nums = to_nums(plaintext)
    c_nums = to_nums(ciphertext)

    # Use only first n blocks to form square matrix
    P = np.array(p_nums[:n*n]).reshape(n, n).T
    C = np.array(c_nums[:n*n]).reshape(n, n).T

    # Compute inverse of P modulo 26
    P_inv = mod_inverse_matrix(P, 26)

    # Recover key
    K = (C.dot(P_inv)) % 26
    return K

# Example key
K = np.array([[3, 3],
              [2, 5]])

plaintext = "HELLOSHIVEN"  # 10 letters
ciphertext = hill_encrypt(plaintext, K, 2)
print("Plain Test:", plaintext)
print("Ciphertext:", ciphertext)

recovered_K = hill_attack(plaintext, ciphertext, 2)
print("Recovered K:\n", recovered_K)
```

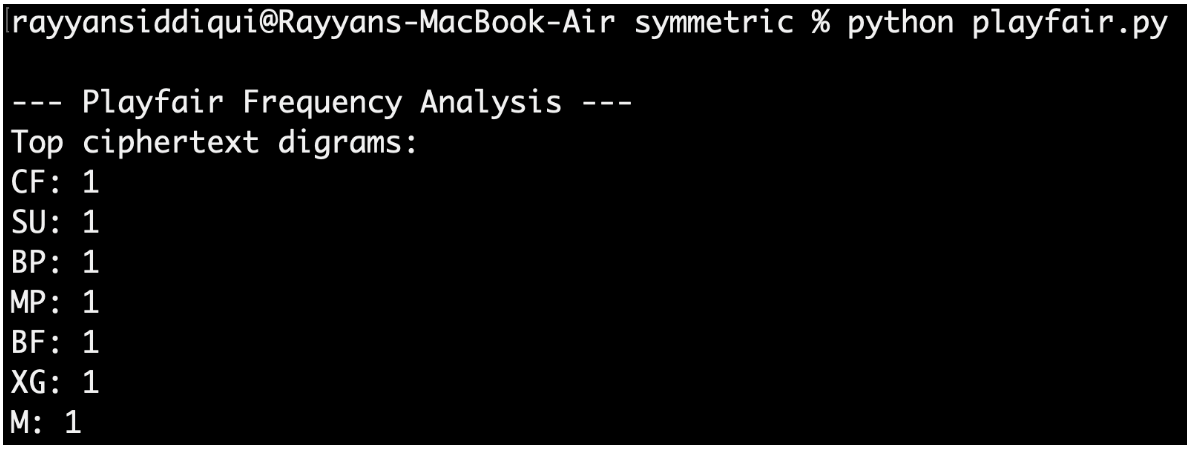
Output



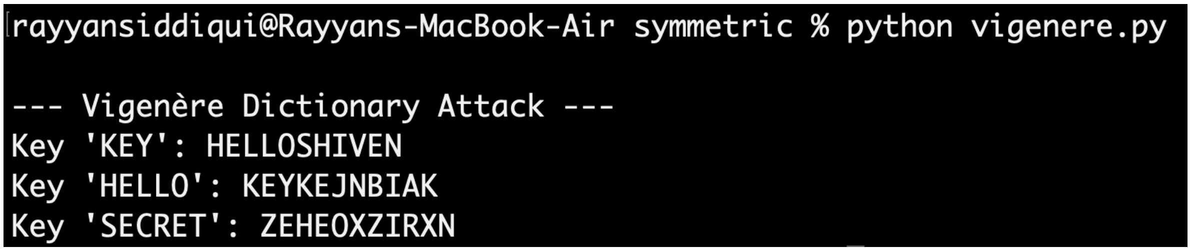
```
[rayyansiddiqui@Rayyans-MacBook-Air symmetric % python part1.py
=== Part 1: Substitution Techniques ===
Caesar: RayyanSiddiqui -> UDBBDQVLGGLTXL -> RAYYANSIDDIQUI

Monoalphabetic: RayyanSiddiqui -> YRSSRXBCMMCOVC -> RAYYANSIDDIQUI

Playfair: RayyanSiddiqui -> MRBWBNAQKBBKLWSA

Hill: RayyanSiddiqui -> ZIOMNNAYSVUSGC -> RAYYANSIDDIQUI

Vigenere: RayyanSiddiqui -> BEWIELCMBNMOEM -> RAYYANSIDDIQUI
```



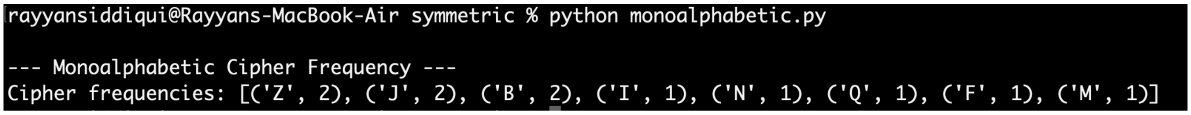
```
[rayyansiddiqui@Rayyans-MacBook-Air symmetric % python caesar.py

--- Caesar Cipher Brute Force ---
Key 0: KHOORVKLYHQ
Key 1: JGNNQUJKXGP
Key 2: IFMMPTIJWFO
Key 3: HELLOSHIVEN
Key 4: GDKKNRGHUDM
Key 5: FCJJMQFGTCL
Key 6: EBIILPEFSBK
Key 7: DAHHKODERAJ
Key 8: CZGGJNCDQZI
Key 9: BYFFIMBCPYH
Key 10: AXEEHLABOXG
Key 11: ZWDDGKZANWF
Key 12: YVCCFJYZMVE
Key 13: XUBBEIXYLUD
Key 14: WTAADHWXKTC
Key 15: VSZZCGVWJSB
Key 16: URYYBFUVIRA
Key 17: TQXXAETUHQZ
Key 18: SPWWZDSTGPY
Key 19: ROVVYCRSFOX
Key 20: QNUUXBQRENW
Key 21: PMTTWAPQDMV
Key 22: OLSSVZOPCLU
Key 23: NKRRUYNOBKT
Key 24: MJQQTXMNAJS
Key 25: LIPPSWLMZIR
```



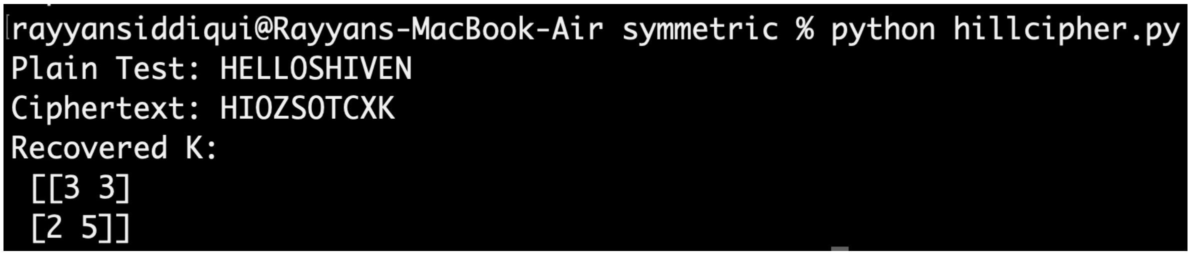
```
[rayyansiddiqui@Rayyans-MacBook-Air symmetric % python playfair.py

--- Playfair Frequency Analysis ---
Top ciphertext digrams:
CF: 1
SU: 1
BP: 1
MP: 1
BF: 1
XG: 1
M: 1
```



```
[rayyansiddiqui@Rayyans-MacBook-Air symmetric % python vigenere.py

--- Vigenère Dictionary Attack ---
Key 'KEY': HELLOSHIVEN
Key 'HELLO': KEYKEJNBIAK
Key 'SECRET': ZEHEOXZIRXN
```



```
[rayyansiddiqui@Rayyans-MacBook-Air symmetric % python monoalphabetic.py

--- Monoalphabetic Cipher Frequency ---
Cipher frequencies: [('Z', 2), ('J', 2), ('B', 2), ('I', 1), ('N', 1), ('Q', 1), ('F', 1), ('M', 1)]
```



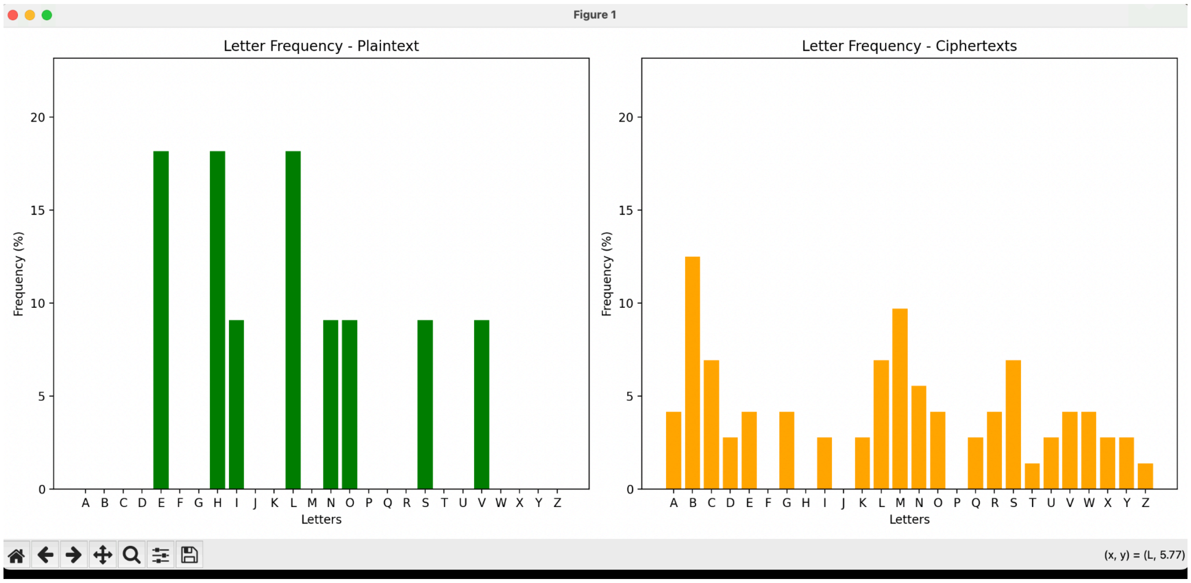
```
[rayyansiddiqui@Rayyans-MacBook-Air symmetric % python hillcipher.py
Plain Test: HELLOSHIVEN
Ciphertext: HIOZSOTCXK
Recovered K:
 [[3 3]
 [2 5]]
```

Part 3

Code

```
import matplotlib.pyplot as plt
from collections import Counter
import string

def get_letter_frequencies(text):
    text = text.upper()
    text = ''.join(filter(str.isalpha, text))
    counter = Counter(text)
    total = sum(counter.values())
    frequencies = {letter: (counter.get(letter, 0) / total) * 100 for letter in
string.ascii_uppercase}
    return frequencies
```

```
plaintext = "RAYYANSIDDIQUI"
ciphertexts = [
    "UDBBDQVLGGLTXL",
    "YRSSRXBCMMCOVC",
    "MRBWBNAQKBBKLWSA",
    "ZIOMNNAYSVUSGC",
    "BEWIELCMBNMOEM"
]


combined_ciphertext = ' '.join(ciphertexts)


plain_freq = get_letter_frequencies(plaintext)
cipher_freq = get_letter_frequencies(combined_ciphertext)


letters = list(string.ascii_uppercase)
plain_values = [plain_freq[letter] for letter in letters]
cipher_values = [cipher_freq[letter] for letter in letters]


plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.bar(letters, plain_values, color='green')
plt.title("Letter Frequency - Plaintext")
plt.xlabel("Letters")
plt.ylabel("Frequency (%)")
plt.ylim(0, max(max(plain_values), max(cipher_values)) + 5)

plt.subplot(1, 2, 2)
plt.bar(letters, cipher_values, color='orange')
plt.title("Letter Frequency - Ciphertexts")
plt.xlabel("Letters")
plt.ylabel("Frequency (%)")
plt.ylim(0, max(max(plain_values), max(cipher_values)) + 5)

plt.tight_layout()
plt.show()
```

Output	 Figure 1 Letter Frequency - Plaintext Frequency (%) 0 5 10 15 20 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z Letters Letter Frequency - Ciphertexts Frequency (%) 0 5 10 15 20 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z Letters (x, y) = (L, 5.77)
Conclusion	In conclusion, symmetric substitution ciphers such as Caesar, Monoalphabetic, Playfair, Hill, and Polyalphabetic form the foundation of classical cryptography, illustrating the evolution of techniques from simple letter shifts to matrix-based encryption. While they effectively demonstrate the core principles of encryption and decryption, their limited key space and predictable patterns make them vulnerable to cryptanalysis and ethical hacking. These experiments highlight both the historical significance and the security limitations of substitution ciphers, emphasizing the need for stronger modern cryptographic methods in real-world applications.