(1) Encryption: • Prompt the user to enter plaintext and a key. • Use the entered key to perform encryption (e.g., Shift cipher). • Display the resulting ciphertext.

(2) Decryption: • Prompt the user to enter ciphertext and the corresponding key used for encryption. • Use the entered key to perform decryption and retrieve the original plaintext. • Display the decrypted plaintext.

(3) Brute Force Attack: • Prompt the user to enter only the ciphertext (without the key). • Implement a brute force attack to try all p

In [ ]:
```python
def encrypt(message, key):
    # Checking if key is a numeric value or not
    if not key.isdigit():
        raise ValueError("Enter numeric value.")
    # Convert the key to an integer for further calculations
    key = int(key)
    # Initialize an empty string to store the encrypted message
    cipher_text = ""
    # Loop through each character in the message
    for character in message:
        # Check if the character is alphanumeric
        if character.isalpha():
            # Determine the base (uppercase or lowercase) for the character
            root = ord('A') if character.isupper() else ord('a')
            # Encrypt the character and append it to the cipher_text
            transformed_character = chr((ord(character) - root + key) % 26 + root)
            cipher_text += transformed_character
        else:
            # If the character is not alphabetic:
            # - Keep it unchanged (e.g., white spaces) as they don't need to be encrypted
            cipher_text += character
    # Return the encrypted message
    return cipher_text


def decrypt_character(character, key):
    # Check if the character is an alphabetical letter
    if character.isalpha():
        # Determine the base (uppercase or lowercase) for the character
        root = ord('A') if character.isupper() else ord('a')
```

```
        # Decrypt the character and return it
        transformed_character = chr((ord(character) - key - root) % 26 + root)
        return transformed_character
    else:
        # If the character is not an alphabet letter, keep it unchanged
        return character

def decrypt(cipher_text, key):
    # Check if the key is a numeric value or not
    if not key.isdigit():
        raise ValueError("Enter numeric value.")
    # Convert the key to an integer for further calculations
    key = int(key)
    # Initialize an empty string to store the decrypted message
    message = ""
    # Loop through each character in the cipher_text
    for character in cipher_text:
        # Decrypt the character and append it to the message
        decrypted_character = decrypt_character(character, key)
        message += decrypted_character
    # Return the decrypted message
    return message

def brute_force_attack(cipher_text):
    # Check if the cipher_text is provided
    if not cipher_text:
        raise ValueError("Provide cipher text.")
    # Check if the cipher_text contains numeric characters
    if any(character.isdigit() for character in cipher_text):
        raise ValueError("Not applicable to numeric cipher.")
    # Initialize a list to store all possible decryption results
    all_results = []
    # Loop through all possible keys (brute force attack)
    for key in range(26):
        # Decrypt the cipher_text with each key and append the result
        decrypted_text = ''.join(decrypt_character(character, key) for character in cipher_text)
        all_results.append(decrypted_text)
    # Return the list of all possible decryption results
    return all_results
```

```
def main():
    print("Select an option:")
    print("1. Encrypt")
    print("2. Decrypt")
    print("3. Brute Force Attack")
    select = input("Choose an option:")

    if select == '1':
         # User input for the message (plaintext)
        message = input("Enter message to be encrypted:")
        # Check if the message is provided
        if not message:
            print("Please provide your message.")
            return
        # Check if the message contains numeric values
        if message.isdigit():
            print("Numeric message not accepted. Please provide alphabetical message")
            return
        # User input for the encryption key
        key = input("Enter your key:")
        try:
            # Attempt to encrypt the message using the provided key
            result = encrypt(message, key)
            # Print the encrypted text (cipher text)
            print("Encrypted text:", result)
        except ValueError as e:
             # Handle the case where an error occurs during encryption (e.g., non-numeric key)
            print(f"Error: {e}")
    elif select == '2':
        # User input for the ciphertext
        cipher_text = input("Enter ciphertext:")
        # Check if the ciphertext is provided
        if not cipher_text:
            print("Please provide the ciphertext.")# if cipher text is nor provided
            return
        # Check if the ciphertext contains numeric values
        if cipher_text.isdigit():
            print("Numeric message not accepted. Please provide alphabetical message")
            return
        # User input for the decryption key
        key = input("Enter the same key as the encryption key:")
```

```
        try:
            # Call the decrypt function and store the result
            result = decrypt(cipher_text, key)
            # Output the decrypted message
            print("Message:", result)
        except ValueError as e:
            # Handle the case where an error occurs during decryption (e.g., non-numeric key)
            print(f"Error: {e}")
    elif select == '3':
        # User input for the ciphertext (encrypted text)
        cipher_text = input("Provide the ciphertext:")
        # Check if the ciphertext is provided
        if not cipher_text:
            print("Provide appropriate ciphertext.")
            return
        # Check if the ciphertext contains numeric values
        if cipher_text.isdigit():
            print("Numeric message not accepted. Please provide alphabetical message")
            return
        try:
            # Call the brute force attack function and store the results in 'all_plaintexts'
            all_plaintexts = brute_force_attack(cipher_text)
            # Print a message indicating the start of the output
            print("\nAll possible combinations of messages with different keys:")
           # Loop through each result in 'all_plaintexts'
            for index, result in enumerate(all_plaintexts, start=0):
                # Print the key and the corresponding decrypted message
                print(f"Key = {index} : {result}")
        except ValueError as e:
             # Handle any ValueError that might be raised during the brute force attack
            print(f"Error: {e}")
    else:
         # Print a message indicating an invalid choice
        print("INVALID")

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