Practical-4

Q1. ImplementVigenère Cipher and Improvised Vigenère Cipher

A1. Vigenère Cipher Overview

The **Vigenère Cipher** is a method of encrypting alphabetic text by using a simple form of polyalphabetic substitution. It uses a keyword, where each letter of the keyword shifts corresponding letters of the plaintext by a number of positions in the alphabet. This cipher is more secure than the Caesar cipher as it uses multiple shifting patterns based on the keyword.

Working of Vigenère Cipher

1. **Encryption**:
    - The plaintext is aligned with the keyword, repeated or truncated as necessary to match its length.
    - Each letter in the plaintext is shifted by the position of the corresponding letter in the keyword (e.g., if the keyword letter is 'B', the plaintext letter is shifted by 1 position).
    - The result is the ciphertext.
2. **Decryption**:
    - The ciphertext is aligned with the keyword.
    - Each letter in the ciphertext is shifted backward by the position of the corresponding letter in the keyword.
    - The result is the original plaintext.

Improved Vigenère Cipher

The **Improved Vigenère Cipher** enhances the traditional Vigenère Cipher by introducing an additional step to modify the keyword:

1. **Key Modification**:
    - The original key is extended to match the length of the plaintext.
    - The key is then rearranged by taking the letters at specific intervals and reversing the sequence, making it harder to predict.

2. **Encryption & Decryption**:
    - The process follows similar steps to the standard Vigenère Cipher but uses the modified key, adding an extra layer of security.

Comparison

- **Standard Vigenère Cipher**: Simpler and easier to implement, but more vulnerable to frequency analysis attacks if the keyword is short or the ciphertext is long.
- **Improved Vigenère Cipher**: Offers enhanced security by making the keyword more complex, making it more resistant to attacks but slightly more complex to implement.

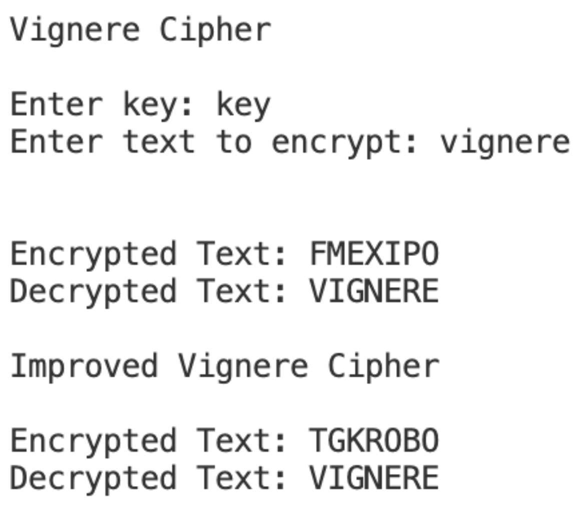
Both methods demonstrate the basic principles of polyalphabetic substitution, providing a foundation for understanding more advanced encryption techniques.

Input:

```
class VignereCipher:

    def __init__(self, key):
        self.key = key  # Initialize the key for the cipher

    def setKey(self, key):
        self.key = key  # Method to set a new key for the cipher

    def encrypt(self, plaintext):
        ciphertext = ""  # Initialize the ciphertext as an empty string
        for i in range(len(plaintext)):
            chNum = ord(plaintext[i]) - ord('A')  # Convert plaintext character to
0-25 range
            chNum += ord(self.key[i % len(self.key)]) - ord('A')  # Add
corresponding key character value
            chNum %= 26  # Wrap around if the sum exceeds 25
            ciphertext += chr(chNum + ord('A'))  # Convert back to a character and
append to ciphertext
        return ciphertext  # Return the encrypted text

    def decrypt(self, ciphertext):
        plaintext = ""  # Initialize the plaintext as an empty string
        for i in range(len(ciphertext)):
            chNum = ord(ciphertext[i]) - ord('A')  # Convert ciphertext character
to 0-25 range
            chNum -= ord(self.key[i % len(self.key)]) - ord('A')  # Subtract
corresponding key character value
            chNum %= 26  # Wrap around if the result is negative
            plaintext += chr(chNum + ord('A'))  # Convert back to a character and
append to plaintext
        return plaintext  # Return the decrypted text


class ImprovedVignereCipher:

    def __init__(self, key):
        self.key = key  # Initialize the key for the improved cipher

    def setKey(self, key):
        self.key = key  # Method to set a new key for the improved cipher

    def encrypt(self, plaintext):
        newKey = ""  # Initialize the new key as an empty string

        # Generate a new key that matches the length of the plaintext
        for i in range(len(plaintext)):
            newKey += self.key[i % len(self.key)]

        # Sort the new key in reverse order
        newKeyOne = ""
        for i in range(len(self.key)):
            j = i
            while j < len(newKey):
                newKeyOne += newKey[j]  # Rearrange the new key in reverse order
                j += len(self.key)

        newKey = ""
        for i in range(len(newKeyOne) - 1, -1, -1):
            newKey += newKeyOne[i]  # Reverse the newKeyOne to form the final
newKey

        ciphertext = ""  # Initialize the ciphertext as an empty string

        # Encrypt using the modified key
        for i in range(len(plaintext)):
            chNum = ord(plaintext[i]) - ord('A')  # Convert plaintext character to
0-25 range
            chNum += ord(newKey[i % len(newKey)]) - ord('A')  # Add corresponding
key character value
            chNum %= 26  # Wrap around if the sum exceeds 25
            ciphertext += chr(chNum + ord('A'))  # Convert back to a character and
append to ciphertext
        return ciphertext  # Return the encrypted text

    def decrypt(self, ciphertext):
        newKey = ""  # Initialize the new key as an empty string

        # Generate a new key that matches the length of the ciphertext
        for i in range(len(ciphertext)):
            newKey += self.key[i % len(self.key)]

        # Sort the new key in reverse order
        newKeyOne = ""
        for i in range(len(self.key)):
            j = i
            while j < len(newKey):
                newKeyOne += newKey[j]  # Rearrange the new key in reverse order
                j += len(self.key)

        newKey = ""
        for i in range(len(newKeyOne) - 1, -1, -1):
            newKey += newKeyOne[i]  # Reverse the newKeyOne to form the final
newKey

        plaintext = ""  # Initialize the plaintext as an empty string

        # Decrypt using the modified key
        for i in range(len(ciphertext)):
            chNum = ord(ciphertext[i]) - ord('A')  # Convert ciphertext character
to 0-25 range
            chNum -= ord(newKey[i % len(newKey)]) - ord('A')  # Subtract
corresponding key character value
            chNum %= 26  # Wrap around if the result is negative
            plaintext += chr(chNum + ord('A'))  # Convert back to a character and
append to plaintext

        return plaintext  # Return the decrypted text


# Main code to demonstrate the Vignere Cipher and Improved Vignere Cipher
print("Vignere Cipher\n")
key = input("Enter key: ")  # Get the key from user input
key = key.upper().replace(" ", "")  # Convert key to uppercase and remove spaces
vignereCipher = VignereCipher(key)  # Create a VignereCipher object
text = input("Enter text to encrypt: ")  # Get the plaintext from user input
text = text.upper().replace(" ", "")  # Convert plaintext to uppercase and remove
spaces
encryptedText = vignereCipher.encrypt(text)  # Encrypt the plaintext
print(f"\n\nEncrypted Text: {encryptedText}")  # Display the encrypted text
decryptedText = vignereCipher.decrypt(encryptedText)  # Decrypt the encrypted text
print(f"Decrypted Text: {decryptedText}")  # Display the decrypted text

print("\nImproved Vignere Cipher\n")
improvedVignereCipher = ImprovedVignereCipher(key)  # Create an
ImprovedVignereCipher object
encryptedText = improvedVignereCipher.encrypt(text)  # Encrypt the plaintext using
the improved cipher
print(f"Encrypted Text: {encryptedText}")  # Display the encrypted text from the
improved cipher
```

```
decryptedText = improvedVignereCipher.decrypt(encryptedText)  # Decrypt the encrypted text from the improved cipher
print(f"Decrypted Text: {decryptedText}\n\n")  # Display the decrypted text from the improved cipher
```

Output:



```
Vignere Cipher

Enter key: key
Enter text to encrypt: vignere


Encrypted Text: FMEXIPO
Decrypted Text: VIGNERE

Improved Vignere Cipher

Encrypted Text: TGKROBO
Decrypted Text: VIGNERE
```