```
16)def printString(S, N):
    Plaintext = [None] * 5
    Freq = [0] * 26
    freqSorted = [None] * 26
    used = [0] * 26
    for I in range(N):
        if S[i] != ‘ ‘:
            freq[ord(S[i]) – 65] += 1
    for I in range(26):
        freqSorted[i] = freq[i]
    T = “ETAOINSHRDLCUMWFGYPBVKJXQZ”
    freqSorted.sort(reverse = True)
    for I in range(5):
        ch = -1
        for j in range(26):
            if freqSorted[i] == freq[j] and used[j] == 0:
                used[j] = 1
                ch = j
                break

        if ch == -1:
            break
        x = ord(T[i]) – 65
        x = x – ch
        curr = “”
        for k in range(N):
            if S[k] == ‘ ‘:
                curr += “ “
                continue
            y = ord(S[k]) – 65
            y += x


            if y < 0:
                y += 26
            if y > 25:
                y -= 26
            curr += chr(y + 65)


        plaintext[i] = curr


    for I in range(5):
        print(plaintext[i])
S = “B TJNQMF NFTTBHF”
N = len(S)


printString(S, N)
```

```
20)cipher_text = “53‡‡†305))6*;4826)4‡.)4‡);806*;48†8¶60))85;;]8*;:‡8†83
(88)5†;46(;88*96*?;8)‡(;485);5†2:‡(;4956*2(5—
4)8¶8*;4069285);)6†8)4‡‡;1(‡9;48081;8:8‡1;48†85;4)485†528806*81
(‡9;48;(88;4(‡?34;48)4‡;161;:188;‡?;”
Plain_text = “”



Mapping = {
    ‘‡’: ‘a’,
    ‘†’: ‘e’,
    ‘¶’: ‘l’,
    ‘*’: ‘o’,
    ‘(‘: ‘u’,
    ‘)’: ‘y’,
    ‘;’: ‘ ‘,
    ‘—‘: ‘-‘,
    ‘]’: ‘,,
    ‘:’: ‘.,
    ‘4’: ‘t’,
    ‘5’: ‘h’,
    ‘8’: ‘s’,
    ‘3’: ‘r’,
    ‘6’: ‘n’,
    ‘0’: ‘g’,
    ‘2’: ‘m’,
    ‘9’: ‘d’,
    ‘1’: ‘l’,
    ‘(‘: ‘u’,
    ‘?’: ‘p’,
    ‘[‘: ‘b’,
    ‘(‘: ‘u’,
    ‘}’: ‘v’,
    ‘7’: ‘c’,
}

For c in cipher_text:
    If c in mapping:
        Plain_text += mapping[c]
    Else:
        Plain_text += c
```

```
Print(plain_text)
```

```
21) from Crypto.Cipher import DES3
Import os

Def pad(text):
    # Add PKCS#7 padding to the plaintext
    Padding_length = 8 – (len(text) % 8)
    Padding = bytes([padding_length] * padding_length)
    Return text + padding

Def unpad(text):
    # Remove PKCS#7 padding from the plaintext
    Padding_length = text[-1]
    Return text[:-padding_length]

Def encrypt_cbc(plaintext, key):
    # Generate a random initialization vector
    Iv = os.urandom(8)

    # Create the 3DES cipher object and initialize with the key and IV
    Cipher = DES3.new(key, DES3.MODE_CBC, iv)

    # Pad the plaintext and encrypt it in CBC mode using 3DES
    Padded_plaintext = pad(plaintext)
    Ciphertext = cipher.encrypt(padded_plaintext)
    # Prepend the IV to the ciphertext
    Return iv + ciphertext

Def decrypt_cbc(ciphertext, key):
    # Extract the IV from the ciphertext
    Iv = ciphertext[:8]

    # Create the 3DES cipher object and initialize with the key and IV
    Cipher = DES3.new(key, DES3.MODE_CBC, iv)

    # Decrypt the ciphertext in CBC mode using 3DES and remove the padding
    Padded_plaintext = cipher.decrypt(ciphertext[8:])
    Plaintext = unpad(padded_plaintext)

    Return plaintext

# Define the plaintext message
Plaintext = b”meet me at the usual place at ten rather than eight oclock”

# Define the initial key
Key =
b”\x01\x23\x45\x67\x89\xAB\xCD\xEF\xFE\xDC\xBA\x98\x76\x54\x32\x10\x01\x23\x45\x67\x89\xAB\x
CD\xEF”

# Encrypt the plaintext message using CBC mode with 3DES
Ciphertext = encrypt_cbc(plaintext, key)

# Decrypt the ciphertext message using CBC mode with 3DES
Decrypted_plaintext = decrypt_cbc(ciphertext, key)
```

```
Print(f”Plaintext: {plaintext}”)

Print(f”Ciphertext: {ciphertext}”)

Print(f”Decrypted plaintext: {decrypted_plaintext}”)
```