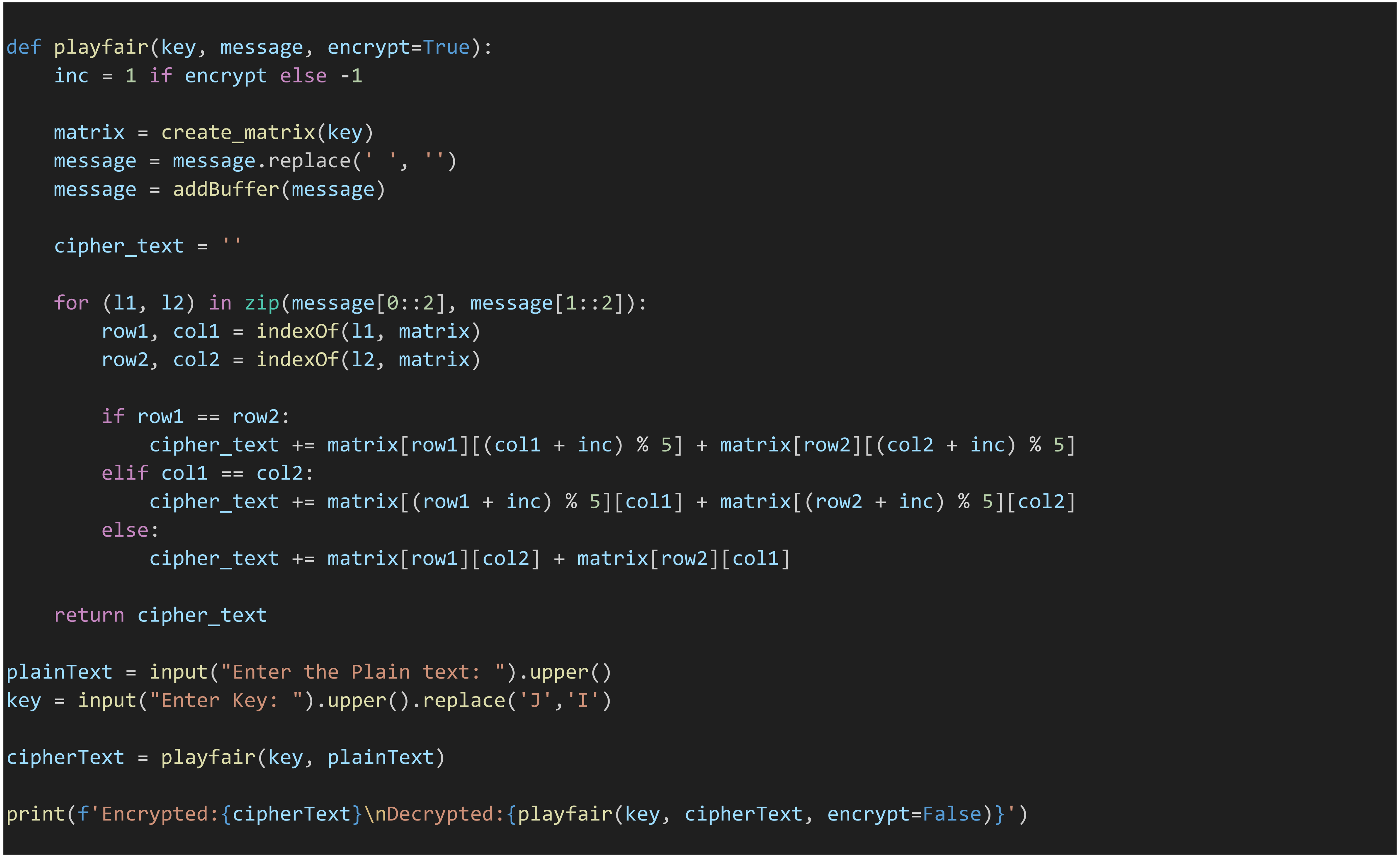
Caesar Cipher:

```python
def encrypt(text,s):
    result = ""
    for char in text:
        if char.isnumeric():
            base = ord('0')
            result += chr((ord(char)-base + s)%10+base)
        else:
            base = ord('A') if char.isupper() else ord('a')
            result += chr((ord(char)-base + s)%26+base)
    return result

def decrypt(text,s):
    result = ""
    for char in text:
        if char.isnumeric():
            base = ord('0')
            result += chr((ord(char)-base - s)%10+base)
        else:
            base = ord('A') if char.isupper() else ord('a')
            result += chr((ord(char)-base - s)%26+base)
    return result

text = "BtechZ9"
s = 4
cipherText=encrypt(text,s)

print (f"Text:{text}\nShift:{s}\nCipher:{cipherText}\nDecrypted:{decrypt(cipherText,s)}")
```

PlayFair cipher:

```python
def create_matrix(key,
list1=['A','B','C','D','E','F','G','H','I','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']):
    temp,matrix = [],[]
    for i in key:
        if i not in temp:
            temp.append(i)
    for i in list1:
        if i not in temp:
            temp.append(i)
    while temp != []:
        matrix.append(temp[:5])
        temp = temp[5:]
        # for i in range(5): temp.pop(0)
    return matrix

def addBuffer(message):
    for index in range(0, len(message)-1, 2):
        l1,l2 =message[index], message[index + 1]
        if l1 == l2:
            message = message[:index + 1] + "X" + message[index + 1:]
    if len(message)%2!=0:
        message+='X'
    return message

def indexOf(letter, matrix):
    for i in range(5):
        try:
            index = matrix[i].index(letter)
            return (i, index)
        except:
            continue
```



```python
def playfair(key, message, encrypt=True):
    inc = 1 if encrypt else -1

    matrix = create_matrix(key)
    message = message.replace(' ', '')
    message = addBuffer(message)

    cipher_text = ''

    for (l1, l2) in zip(message[0::2], message[1::2]):
        row1, col1 = indexOf(l1, matrix)
        row2, col2 = indexOf(l2, matrix)

        if row1 == row2:
            cipher_text += matrix[row1][(col1 + inc) % 5] + matrix[row2][(col2 + inc) % 5]
        elif col1 == col2:
            cipher_text += matrix[(row1 + inc) % 5][col1] + matrix[(row2 + inc) % 5][col2]
        else:
            cipher_text += matrix[row1][col2] + matrix[row2][col1]

    return cipher_text

plainText = input("Enter the Plain text: ").upper()
key = input("Enter Key: ").upper().replace('J','I')

cipherText = playfair(key, plainText)

print(f'Encrypted:{cipherText}\nDecrypted:{playfair(key, cipherText, encrypt=False)}')
```

Vigenere Cipher:

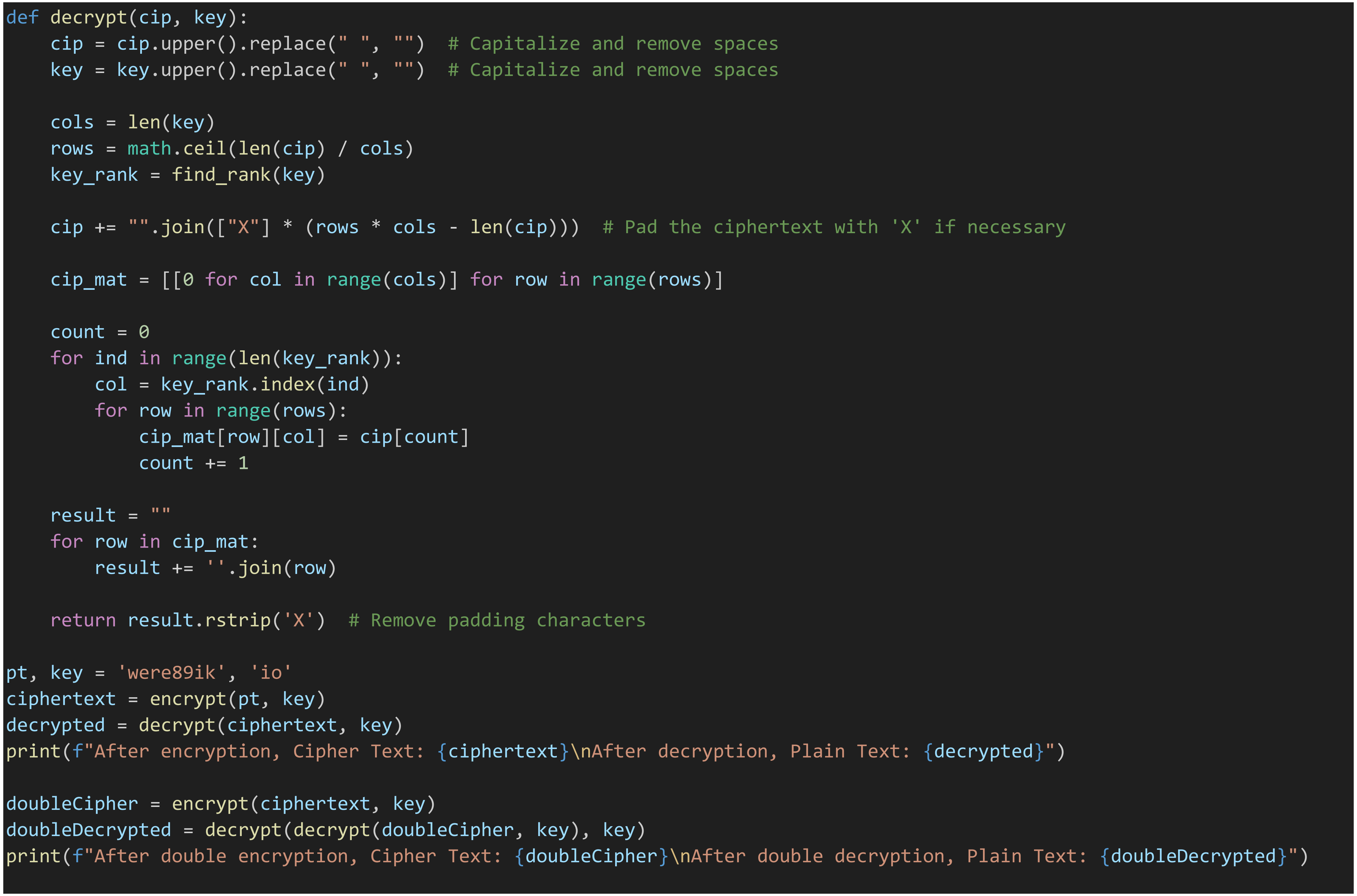
```python
plain_txt = input("Enter plaintext : ").upper()
key = input("Enter Key : ").upper()
padd_key = key

if len(plain_txt) > len(key):
    for i in range(len(plain_txt)-len(key)):
        padd_key += key[i%len(key)]

encrpyted,decrpyted = "",""

for i in range(len(plain_txt)):
    encrpyted += chr(((ord(plain_txt[i]) + ord(padd_key[i])) % 26) + 65)
for i in range(len(plain_txt)):
    decrpyted += chr(((ord(encrpyted[i]) - ord(padd_key[i])) % 26) + 65)

print(f"\nPlain Text : {plain_txt}\nKey : {key}\nPadded Key : {padd_key}")
print(f"\nafter encrption, cipher text : {encrpyted}\nafter decyprtion, decrypted text : {decrpyted}")
```

Vernam cipher:

```
plain = input("Enter text: ").upper()
key = input("Enter key: ").upper()
padded_key = key

if len(key)<len(plain):
    for i in range(len(plain)-len(key)):
        padded_key+=key[i%len(key)]

key = padded_key

encrypt, decrypt = "",""

for p,q in zip(plain,key):
    encrypt += chr(ord(p)^ord(q))

for p,q in zip(encrypt,key):
    decrypt += chr(ord(p)^ord(q))

print("Encrypted: ",encrypt)
print("Decrypted: ",decrypt)
```

Columnar Cipher:

```python
import math

def find_rank(key):
    rank = 0
    for i in sorted(key):
        key = key.replace(i, str(rank), 1)
        rank += 1
    key = [int(i) for i in key]
    return key

def encrypt(pt, key):
    pt = pt.upper().replace(" ", "")  # Capitalize and remove spaces
    key = key.upper().replace(" ", "")  # Capitalize and remove spaces

    cols = len(key)
    rows = math.ceil(len(pt) / cols)
    key_rank = find_rank(key)
    print(key_rank)

    pt += "".join(["X"] * (rows * cols - len(pt)))  # Pad the plaintext with 'X'

    matrix = [list(pt[i: i+cols]) for i in range(0, len(pt), cols)]
    for i in range(rows):
        print(matrix[i])

    ciphertext = ""
    for ind in range(len(key_rank)):
        col = key_rank.index(ind)
        for i in range(rows):
            ciphertext += matrix[i][col]

    return ciphertext
```



```python
def decrypt(cip, key):
    cip = cip.upper().replace(" ", "")  # Capitalize and remove spaces
    key = key.upper().replace(" ", "")  # Capitalize and remove spaces

    cols = len(key)
    rows = math.ceil(len(cip) / cols)
    key_rank = find_rank(key)

    cip += "".join(["X"] * (rows * cols - len(cip)))  # Pad the ciphertext with 'X' if necessary

    cip_mat = [[0 for col in range(cols)] for row in range(rows)]

    count = 0
    for ind in range(len(key_rank)):
        col = key_rank.index(ind)
        for row in range(rows):
            cip_mat[row][col] = cip[count]
            count += 1

    result = ""
    for row in cip_mat:
        result += ''.join(row)

    return result.rstrip('X')  # Remove padding characters

pt, key = 'were89ik', 'io'
ciphertext = encrypt(pt, key)
decrypted = decrypt(ciphertext, key)
print(f"After encryption, Cipher Text: {ciphertext}\nAfter decryption, Plain Text: {decrypted}")

doubleCipher = encrypt(ciphertext, key)
doubleDecrypted = decrypt(decrypt(doubleCipher, key), key)
print(f"After double encryption, Cipher Text: {doubleCipher}\nAfter double decryption, Plain Text: {doubleDecrypted}")
```

RSA and Digital Signature:

```
#RSA encryption/Decryption and Digital Signature
import random, hashlib

def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def generate_keys():
    p = int(input("Enter p (greater than 100): ")) #113
    q = int(input("Enter q (greater than 100): ")) #101
    n = p * q
    phi = (p - 1) * (q - 1)
    e = random.randint(1, phi)
    while gcd(e, phi) != 1:
        e = random.randint(1, phi)
    d = pow(e, -1, phi)
    return (e, n), (d, n)

def encrypt(message, public_key):
    e, n = public_key
    encrypted_message = [pow(ord(char), e, n) for char in message]
    return encrypted_message

def decrypt(encrypted_message, private_key):
    d, n = private_key
    decrypted_message = [chr(pow(char, d, n)) for char in encrypted_message]
    return ''.join(decrypted_message)

message = "Hello, World!"
public_key, private_key = generate_keys()
```

```python
encrypted_message = encrypt(message, public_key)
decrypted_message = decrypt(encrypted_message, private_key)

print("Public key: ",public_key)
print("Private key: ",private_key)

print(f"Original message:{message}\nEncryptedMeessage:{encrypted_message}\nDecryptedMsg:{decrypted_message}")

#Digital Signature part
hash_value = hashlib.md5(message.encode()).hexdigest()
print("Hash value at sender end: ", hash_value)

signature = encrypt(hash_value, public_key)
print("Digital signature: ", signature)

# receiver side
hash_value_check = decrypt(signature, private_key)
print("Hash value checked at receiver end: ", hash_value_check)

if hash_value_check == hash_value:
    print("Verified")
else:
    print("Not verified")
```

Diffie-Hellman:

```python
p = 23
g = 7
a = 4
b = 3

if a>=p or b>=p:
    print("a or b is less than p")
else:
    XA = (g**a) % p
    XB = (g**b) % p
    AK = (XB**a) % p
    BK = (XA**b) % p

    if AK==BK:
        print(AK,BK,"User verified")
    else:
        print("nahhh")
```