DOP: / /2023 **DOS: / /2023**

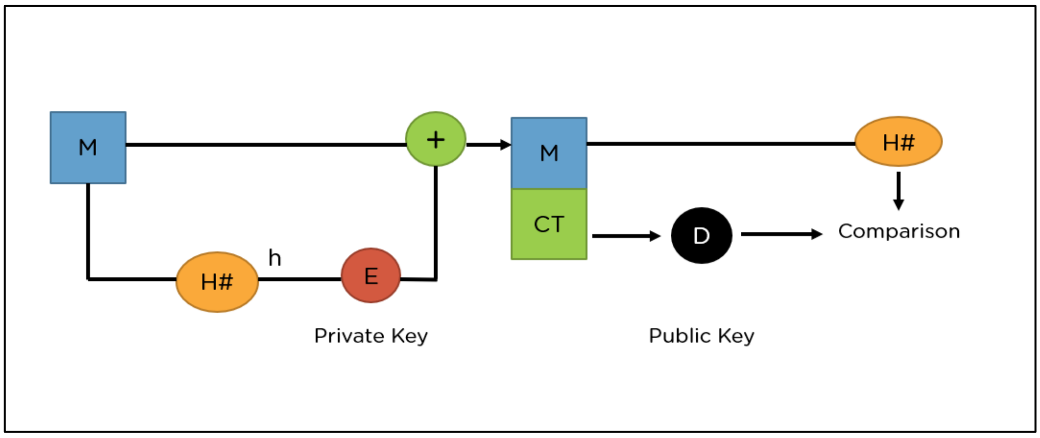
Experiment No: 06

Aim: Implementation and analysis of RSA cryptosystem and Digital signature scheme using RSA.

Theory:

🔷RSA algorithm:

The RSA algorithm is a public-key signature algorithm developed by Ron Rivest, Adi Shamir, and Leonard Adleman. Their paper was first published in 1977, and the algorithm uses logarithmic functions to keep the working complex enough to withstand brute force and streamlined enough to be fast post-deployment. The image below shows it verifies the digital signatures using RSA methodology.



🔷RSA in Data Encryption

When using RSA for encryption and m decryption of general data, it reverses the key set usage. Unlike signature verification, it uses the receiver's public key to encrypt the data, and it uses the receiver's private key in decrypting the data. Thus, there is no need to exchange any keys in this scenario.

There are two broad components when it comes to RSA cryptography, they are:

- Key Generation: Generating the keys to be used for encrypting and decrypting the data to be exchanged.
- Encryption/Decryption Function: The steps that need to be run when scrambling and recovering the data.

You will now understand each of these steps in our next sub-topic.

🔷Steps in RSA Algorithm:

Keeping the image above in mind, go ahead and see how the entire process works, starting from creating the key pair, to encrypting and decrypting the information.

Key Generation

You need to generate public and private keys before running the functions to generate your ciphertext and plaintext. They use certain variables and parameters, all of which are explained below:

- Choose two large prime numbers (p and q)
- Calculate n = p*q and z = (p-1)(q-1)
- Choose a number e where 1 < e < z
- Calculate d = e-1mod(p-1)(q-1)
- You can bundle private key pair as (n,d)
- You can bundle public key pair as (n,e)

Encryption/Decryption Function

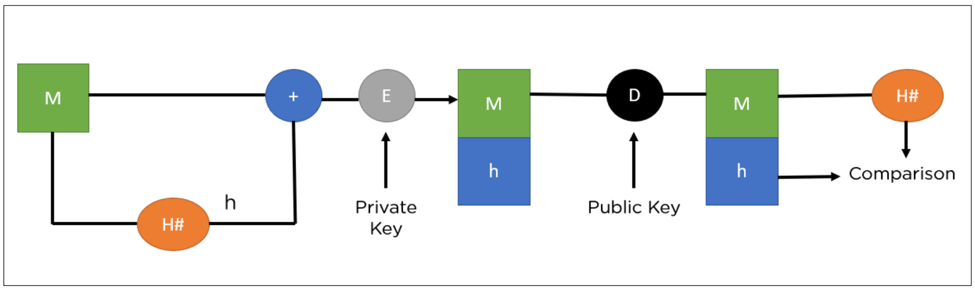
Once you generate the keys, you pass the parameters to the functions that calculate your ciphertext and plaintext using the respective key.

- If the plaintext is m, ciphertext = me mod n.
- If the ciphertext is c, plaintext = cd mod n

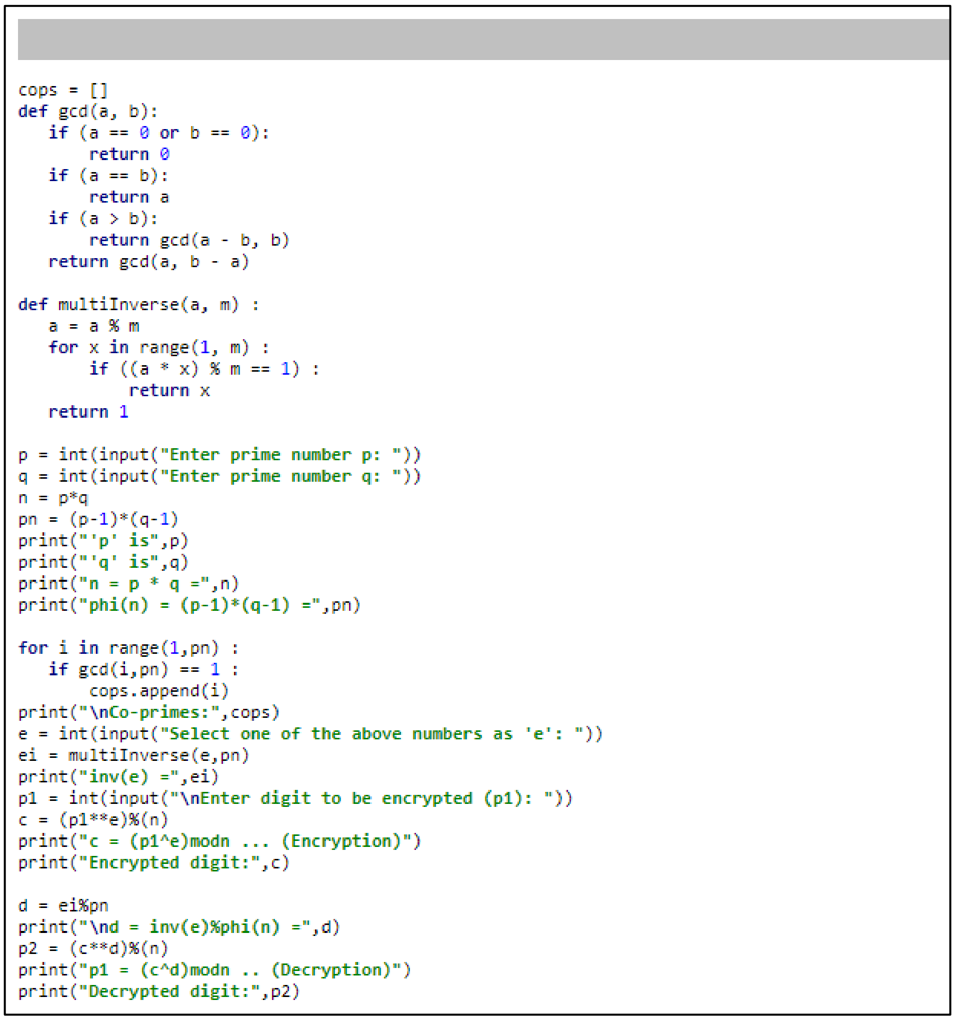
🔷What Are Digital Signatures?

Digital signatures serve the purpose of authentication and verification of documents and files. This is crucial to prevent tampering during official papers' transmission and prevent digital manipulation or forgery.

They work on the public key cryptography architecture, barring one small caveat. Typically, the asymmetric key system uses a public key for encryption and a private key for decryption. However, when dealing with digital signatures, it's the opposite. The private key is used to encrypt the signature, and the public key is used to decrypt it. Since the keys work in tandem with each other, decrypting it with the public key signifies it used the correct private key to sign the document, hence authenticating the origin of the signature.

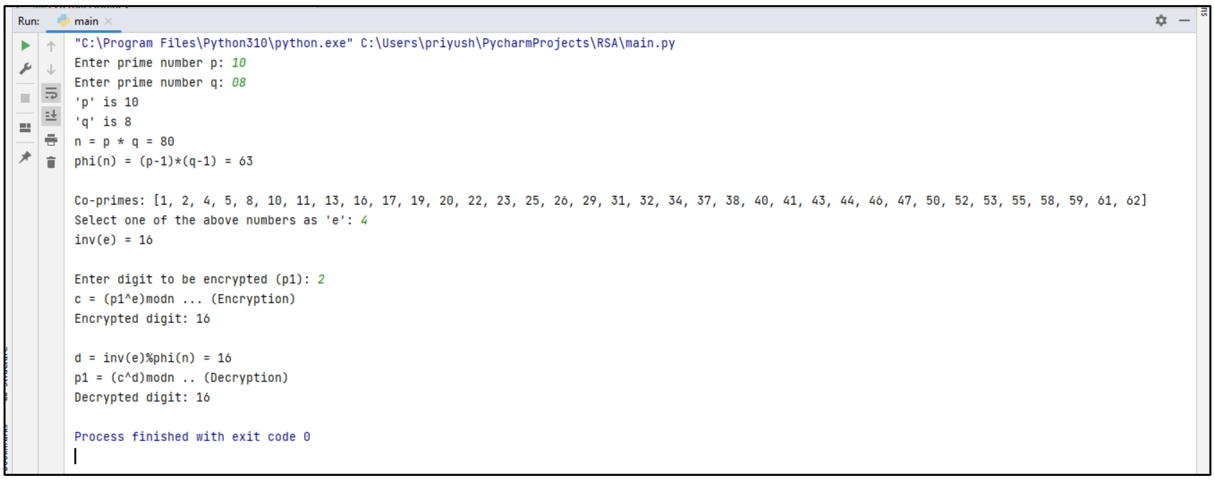


Input:



```
cops = []
def gcd(a, b):
   if (a == 0 or b == 0):
       return 0
   if (a == b):
       return a
   if (a > b):
       return gcd(a - b, b)
   return gcd(a, b - a)

def multiInverse(a, m) :
   a = a % m
   for x in range(1, m) :
       if ((a * x) % m == 1) :
           return x
   return 1

p = int(input("Enter prime number p: "))
q = int(input("Enter prime number q: "))
n = p*q
pn = (p-1)*(q-1)
print("'p' is",p)
print("'q' is",q)
print("n = p * q =",n)
print("phi(n) = (p-1)*(q-1) =",pn)

for i in range(1,pn) :
   if gcd(i,pn) == 1 :
       cops.append(i)
print("\nCo-primes:",cops)
e = int(input("Select one of the above numbers as 'e': "))
ei = multiInverse(e,pn)
print("inv(e) =",ei)
p1 = int(input("\nEnter digit to be encrypted (p1): "))
c = (p1**e)%(n)
print("c = (p1^e)modn ... (Encryption)")
print("Encrypted digit:",c)

d = ei%pn
print("\nd = inv(e)%phi(n) =",d)
p2 = (c**d)%(n)
print("p1 = (c^d)modn .. (Decryption)")
print("Decrypted digit:",p2)
```

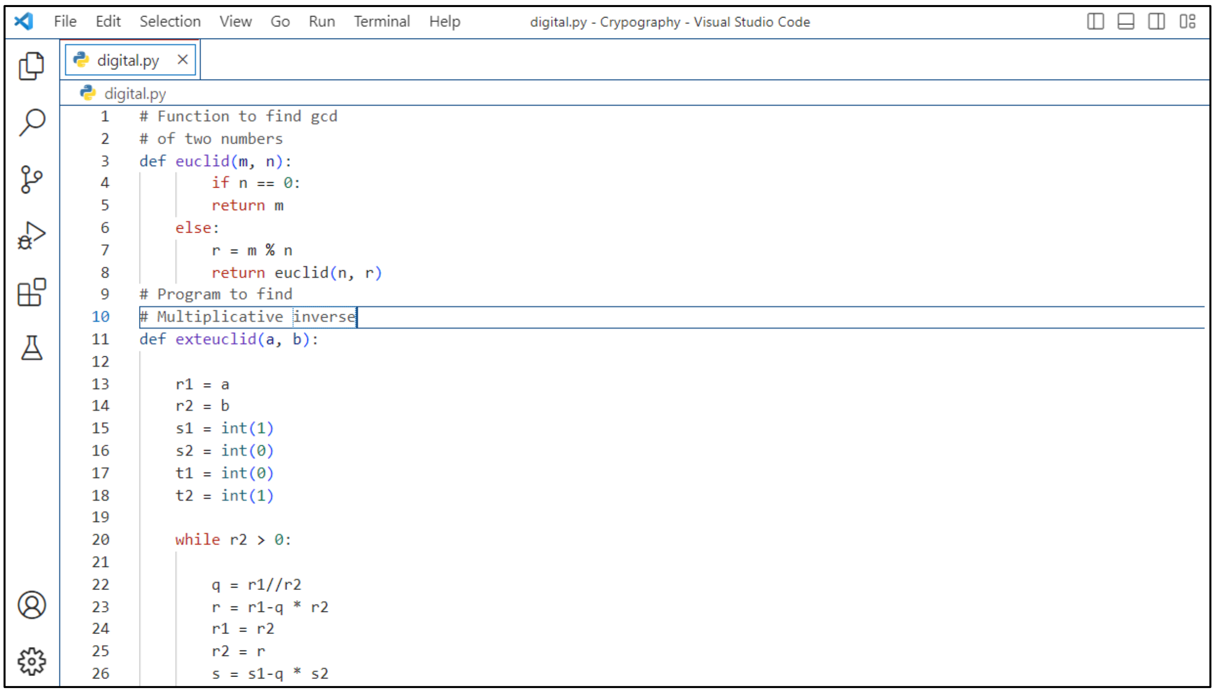
Output:



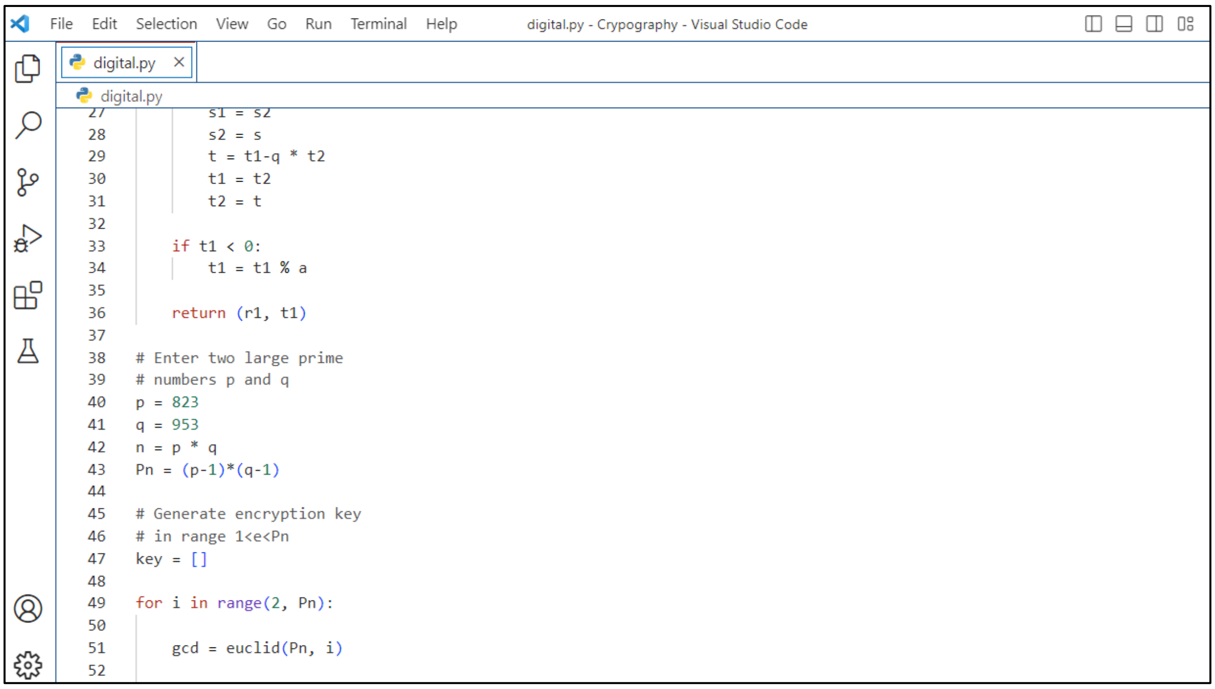
```
"C:\Program Files\Python310\python.exe" C:\Users\priyush\PycharmProjects\RSA\main.py
Enter prime number p: 10
Enter prime number q: 08
'p' is 10
'q' is 8
n = p * q = 80
phi(n) = (p-1)*(q-1) = 63

Co-primes: [1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20, 22, 23, 25, 26, 29, 31, 32, 34, 37, 38, 40, 41, 43, 44, 46, 47, 50, 52, 53, 55, 58, 59, 61, 62]
Select one of the above numbers as 'e': 4
inv(e) = 16

Enter digit to be encrypted (p1): 2
c = (p1^e)modn ... (Encryption)
Encrypted digit: 16

d = inv(e)%phi(n) = 16
p1 = (c^d)modn .. (Decryption)
Decrypted digit: 16

Process finished with exit code 0
```

Digital signature scheme using RSA.

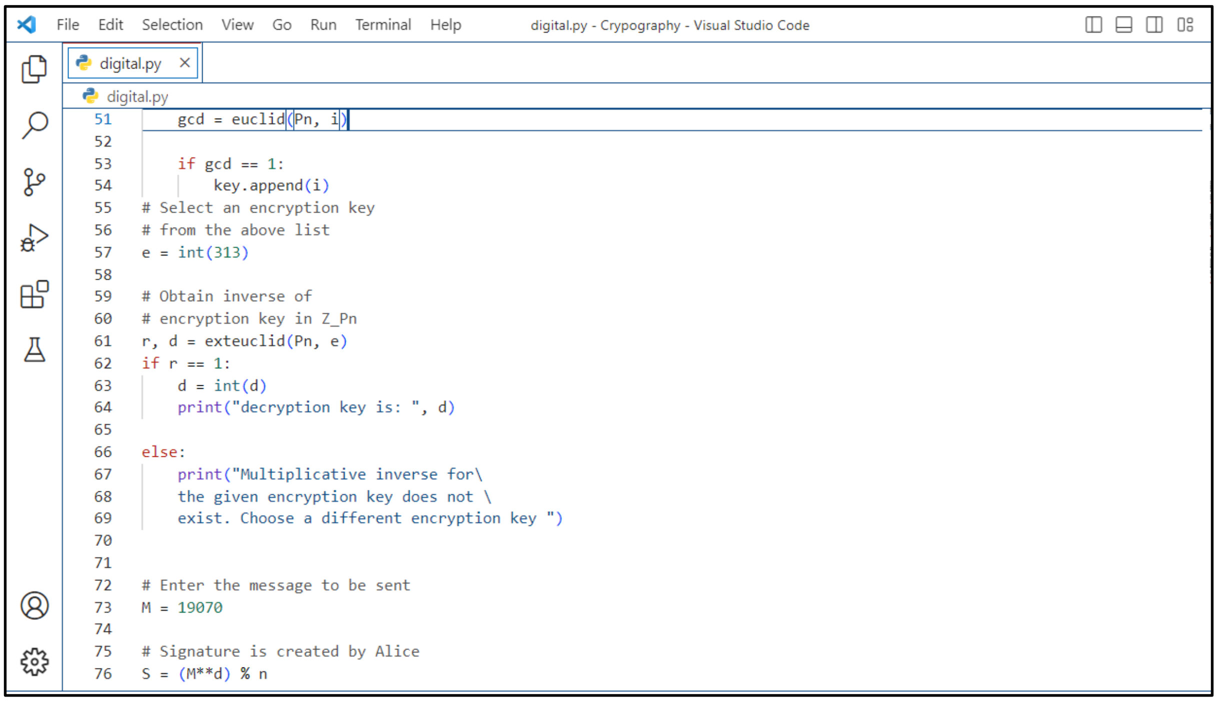
Input:


File Edit Selection View Go Run Terminal Help
digital.py - Crypography - Visual Studio Code
digital.py
digital.py

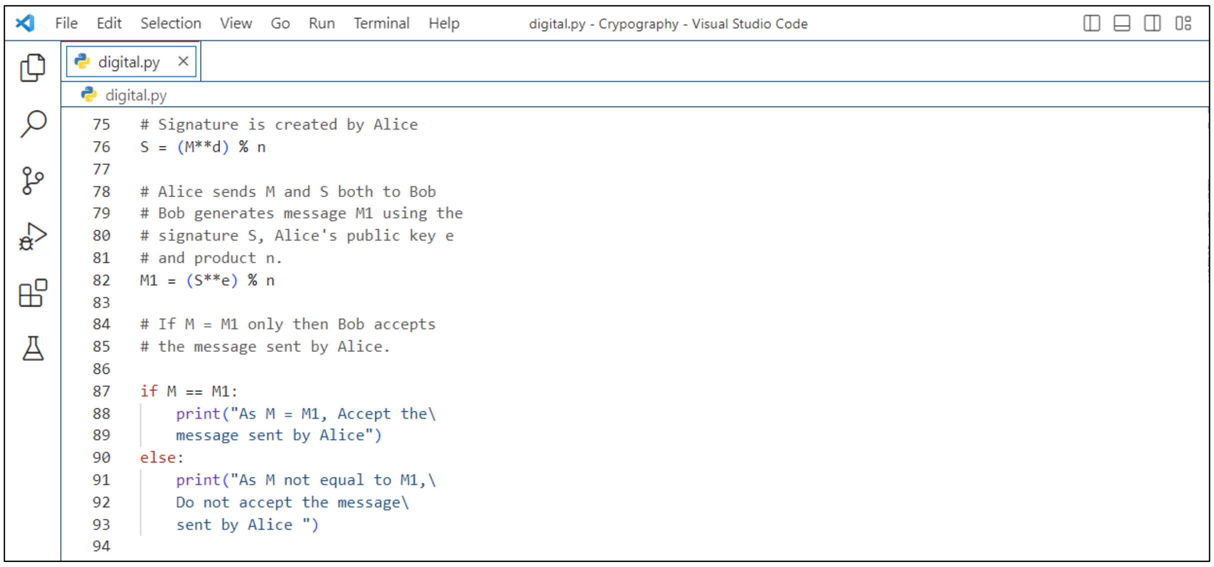
```
# Function to find gcd
# of two numbers
def euclid(m, n):
        if n == 0:
        return m
    else:
        r = m % n
        return euclid(n, r)
# Program to find
# Multiplicative inverse
def exteuclid(a, b):

    r1 = a
    r2 = b
    s1 = int(1)
    s2 = int(0)
    t1 = int(0)
    t2 = int(1)

    while r2 > 0:

        q = r1//r2
        r = r1-q * r2
        r1 = r2
        r2 = r
        s = s1-q * s2
```


File Edit Selection View Go Run Terminal Help
digital.py - Crypography - Visual Studio Code
digital.py

```
            s1 = s2
            s2 = s
            t = t1-q * t2
            t1 = t2
            t2 = t

        if t1 < 0:
            t1 = t1 % a

        return (r1, t1)

    # Enter two large prime
    # numbers p and q
    p = 823
    q = 953
    n = p * q
    Pn = (p-1)*(q-1)

    # Generate encryption key
    # in range 1<e<Pn
    key = []

    for i in range(2, Pn):

        gcd = euclid(Pn, i)

```

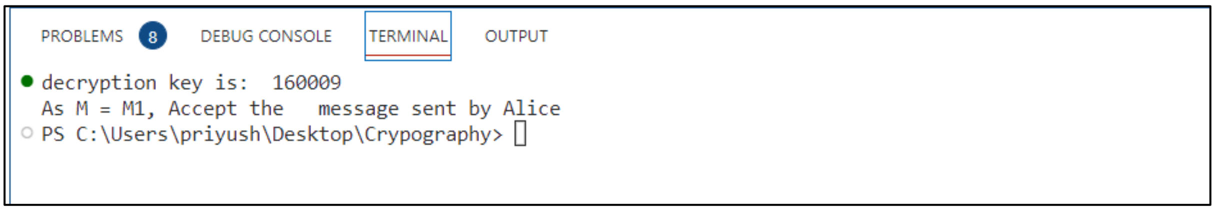

File Edit Selection View Go Run Terminal Help
digital.py - Crypography - Visual Studio Code
digital.py

```
        gcd = euclid(Pn, i)

        if gcd == 1:
            key.append(i)
    # Select an encryption key
    # from the above list
    e = int(313)

    # Obtain inverse of
    # encryption key in Z_Pn
    r, d = exteuclid(Pn, e)
    if r == 1:
        d = int(d)
        print("decryption key is: ", d)

    else:
        print("Multiplicative inverse for\
        the given encryption key does not \
        exist. Choose a different encryption key ")


    # Enter the message to be sent
    M = 19070

    # Signature is created by Alice
    S = (M**d) % n
```


File Edit Selection View Go Run Terminal Help
digital.py - Crypography - Visual Studio Code
digital.py

```python
# Signature is created by Alice
S = (M**d) % n

# Alice sends M and S both to Bob
# Bob generates message M1 using the
# signature S, Alice's public key e
# and product n.
M1 = (S**e) % n

# If M = M1 only then Bob accepts
# the message sent by Alice.

if M == M1:
    print("As M = M1, Accept the\
    message sent by Alice")
else:
    print("As M not equal to M1,\
    Do not accept the message\
    sent by Alice ")
```

Output:



```
decryption key is:  160009
As M = M1, Accept the    message sent by Alice
PS C:\Users\priyush\Desktop\Crypography>
```

🔷**Advantages of RSA:**

- **No Key Sharing**: RSA encryption depends on using the receiver's public key, so you don't have to share any secret key to receive messages from others.
- **Proof of Authenticity**: Since the key pairs are related to each other, a receiver can't intercept the message since they won't have the correct private key to decrypt the information.
- **Faster Encryption**: The encryption process is faster than that of the DSA algorithm.
- **Data Can't Be Modified**: Data will be tamper-proof in transit since meddling with the data will alter the usage of the keys. And the private key won't be able to decrypt the information, hence alerting the receiver of manipulation.

Conclusion: - Thus we Implementation and analysis of RSA cryptosystem and Digital signature scheme using RSA.