COMSATS University Islamabad, Attock campus
Department of Computer Science

Program: BS-SE

Assignment#04

Registration NO: Sp24-BSE-052

Name: Eman Ejaz

Date: 2nd-December -2025

Subject: Information security Lab

Submitted to: Ma'am Ambreen GUL

Task 1:

understanding how RSA math works.

Explanation:

The program demonstrates a **basic implementation of the RSA algorithm** using small prime numbers. Two prime numbers are selected to generate the value of **n** and **ϕ(n)**, which are used to create the **public key (e, n)** and the **private key (d, n)**.

The **encrypt function** converts each character of the message into its numeric ASCII value and then applies the RSA formula C = M^e mod n to produce the ciphertext.

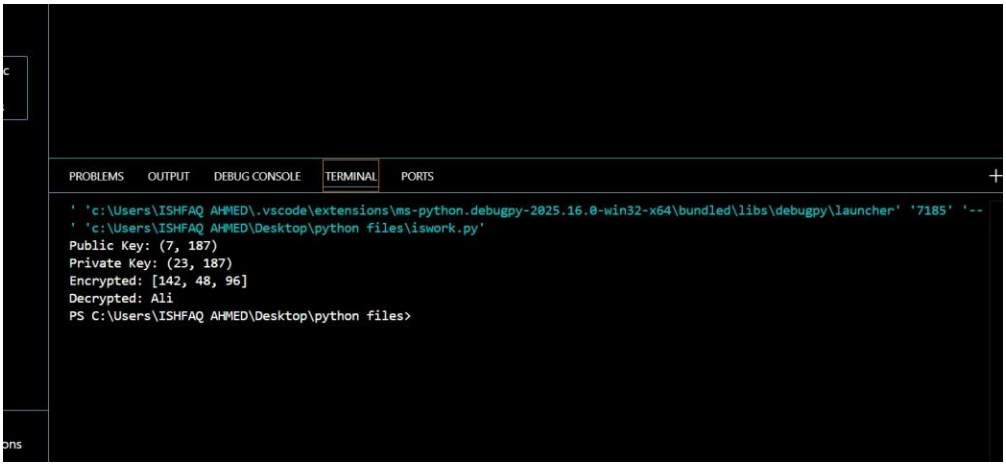
The **decrypt function** uses the private key and the formula M = C^d mod n to convert the encrypted numbers back into the original characters, successfully recovering the plaintext.

This code shows how RSA works mathematically without using any external libraries, helping in understanding the core concept of public key cryptography.

```
# Step 1: Pick two small prime numbers
p = 17
q = 11

# Step 2: Compute n
n = p * q

# Step 3: Euler's Totient
phi = (p - 1) * (q - 1)

# Step 4: Choose public exponent e
e = 7   # Must be coprime with phi

# Step 5: Find private key d
# Solve: e*d ≡ 1 (mod phi)

def mod_inverse(e, phi):
    for d in range(1, phi):
        if (e * d) % phi == 1:
            return d

d = mod_inverse(e, phi)
```

```python
# Public Key: (e, n)
# Private Key: (d, n)
print("Public Key:", (e, n))
print("Private Key:", (d, n))
# ---------- Encryption ----------
def encrypt(message, e, n):
    encrypted = []
    for char in message:
        m = ord(char)
        c = pow(m, e, n)
        encrypted.append(c)
    return encrypted
# ---------- Decryption ----------
def decrypt(cipher, d, n):
    decrypted = ""
    for c in cipher:
        m = pow(c, d, n)
        decrypted += chr(m)
    return decrypted
# Test
msg = "Ali"
ciphertext = encrypt(msg, e, n)
print("Encrypted:", ciphertext)

plaintext = decrypt(ciphertext, d, n)
print("Decrypted:", plaintext)
```

OUTPUT:



Task 2:

Using cryptography tools

Explanation:

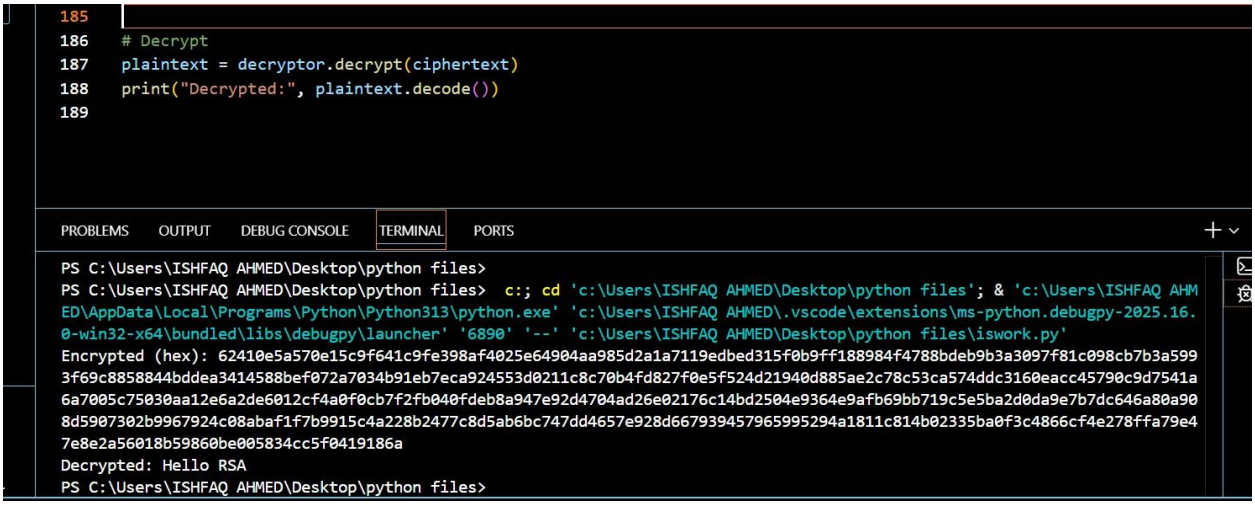
In this task, a real-world cryptographic library is used to perform RSA encryption and decryption. A **2048-bit RSA key pair** is generated using the library, which provides much stronger security than small, manually chosen numbers.

The user enters a message, which is then **encrypted using the public key**. The encrypted data (ciphertext) is displayed in **hexadecimal format** for readability. After that, the **private key** is used to decrypt the ciphertext and recover the original message.

This task demonstrates how modern applications use cryptographic libraries to securely handle real data instead of performing calculations manually.

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

# Generate 2048-bit RSA keys
key = RSA.generate(2048)

public_key = key.publickey()
private_key = key

# Create cipher objects
encryptor = PKCS1_OAEP.new(public_key)
decryptor = PKCS1_OAEP.new(private_key)

message = b"Hello RSA"

# Encrypt
ciphertext = encryptor.encrypt(message)
print("Encrypted (hex):", ciphertext.hex())

# Decrypt
plaintext = decryptor.decrypt(ciphertext)
print("Decrypted:", plaintext.decode())
```

OUTPUT:



Task 3:

Prove integrity + authenticity

Explanation:

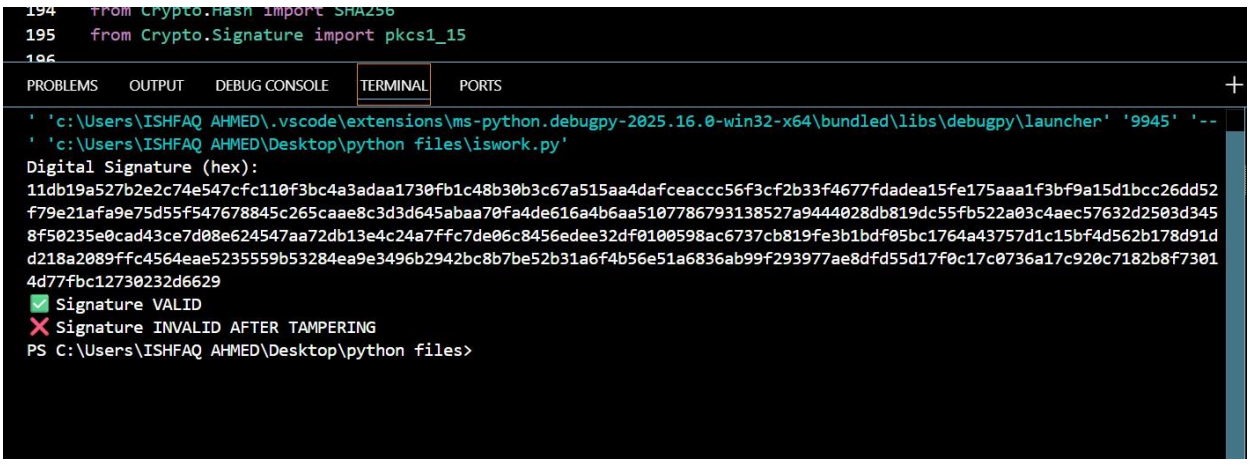
1. An **RSA key pair** is generated for signing and verification.
2. The **message** is hashed using **SHA256**, producing a fixed-length digest.
3. The **private key signs** the hash to create a digital signature.
4. The **public key verifies** the signature against the original message.
5. If the message is **modified**, verification fails, demonstrating data integrity and authenticity.

Digital signatures ensure that a message is **from a legitimate sender** and has **not been altered**.

```
from Crypto.PublicKey import RSA
from Crypto.Hash import SHA256
from Crypto.Signature import pkcs1_15

# ---------- Generate RSA key ----------
sign_key = RSA.generate(2048)
pub_key = sign_key.publickey()

# ---------- Message ----------
message = b"RSA Signature Test"
```

```python
# ---------- Create Hash ----------
hash_obj = SHA256.new(message)

# ---------- Create Signature ----------
signature = pkcs1_15.new(sign_key).sign(hash_obj)
print("Digital Signature (hex):")
print(signature.hex())


# ---------- Verify Original Message ----------
try:
    pkcs1_15.new(pub_key).verify(hash_obj, signature)
    print("✅  Signature VALID")
except:
    print("❌ Signature INVALID")


# ---------- Message Tampering ----------
tampered = b"RSA signature test"  # small change

tampered_hash = SHA256.new(tampered)

try:
    pkcs1_15.new(pub_key).verify(tampered_hash, signature)
    print("✅  Signature VALID")
except:
    print("❌ Signature INVALID AFTER TAMPERING"
```

OUTPUT:


194 from Crypto.Hash import SHA256
195 from Crypto.Signature import pkcs1_15
196
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
' 'c:\Users\ISHFAQ AHMED\.vscode\extensions\ms-python.debugpy-2025.16.0-win32-x64\bundled\libs\debugpy\launcher' '9945' '--
' 'c:\Users\ISHFAQ AHMED\Desktop\python files\iswork.py'
Digital Signature (hex):
11db19a527b2e2c74e547cfc110f3bc4a3adaa1730fb1c48b30b3c67a515aa4dafceaccc56f3cf2b33f4677fdadea15fe175aaa1f3bf9a15d1bcc26dd52
f79e21afa9e75d55f547678845c265caae8c3d3d645abaa70fa4de616a4b6aa5107786793138527a9444028db819dc55fb522a03c4aec57632d2503d345
8f50235e0cad43ce7d08e624547aa72db13e4c24a7ffc7de06c8456edee32df0100598ac6737cb819fe3b1bdf05bc1764a43757d1c15bf4d562b178d91d
d218a2089ffc4564eae5235559b53284ea9e3496b2942bc8b7be52b31a6f4b56e51a6836ab99f293977ae8dfd55d17f0c17c0736a17c920c7182b8f7301
4d77fbc12730232d6629
✅ Signature VALID
❌ Signature INVALID AFTER TAMPERING
PS C:\Users\ISHFAQ AHMED\Desktop\python files>
```
```