

PRACTICAL 10

DATE: 17-04-2025

AIM: Implement RSA Encryption-Decryption in Python.

Code:

```

import random

# Helper: Compute GCD
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

# Helper: Compute modular inverse using Extended Euclidean Algorithm
def modinv(a, m):
    m0, x0, x1 = m, 0, 1
    while a > 1:
        q = a // m
        m, a = a % m, m
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1

# Check for primality (naive, not suitable for large numbers)
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5)+1):
        if n % i == 0:
            return False
    return True

# Generate RSA keys
def generate_keys(p, q):
    if not (is_prime(p) and is_prime(q)):
        raise ValueError('Both numbers must be prime.')
    elif p == q:
        raise ValueError('p and q cannot be equal')

    n = p * q
    phi = (p - 1) * (q - 1)

    # Choose e
    e = 65537 # Common choice
    if gcd(e, phi) != 1:
        # Try finding an alternate e
        e = 3

```

```
while gcd(e, phi) != 1:  
    e += 2  
  
# Calculate d  
d = modinv(e, phi)  
  
return ((e, n), (d, n)) # public key, private key  
  
# Encrypt message  
def encrypt(public_key, plaintext):  
    e, n = public_key  
    cipher = [pow(ord(char), e, n) for char in plaintext]  
    return cipher  
  
# Decrypt message  
def decrypt(private_key, ciphertext):  
    d, n = private_key  
    plain = [chr(pow(char, d, n)) for char in ciphertext]  
    return ''.join(plain)  
  
# Example usage  
if __name__ == '__main__':  
    p = 61  
    q = 53  
    public, private = generate_keys(p, q)  
  
    message = "Hello RSA"  
    print("Original message:", message)  
  
    encrypted = encrypt(public, message)  
    print("Encrypted message:", encrypted)  
  
    decrypted = decrypt(private, encrypted)  
    print("Decrypted message:", decrypted)
```

Output:

```
Original message: Hello RSA  
Encrypted message: [3000, 1313, 745, 745, 2185, 1992, 1859, 2680, 2790]  
Decrypted message: Hello RSA
```

Code:

The screenshot shows a code editor interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** Search icon.
- Toolbars:** Explorer, Outline, Timeline, Maven.
- Code Area:** The file `RSA_Encryption_Decryption.py` is open. The code implements RSA encryption and decryption, including helper functions for GCD and modular inverses, and a primality check. It also includes a function to generate RSA keys from prime numbers p and q .

```
File Edit Selection View Go Run Terminal Help ⏪ ⏴ Search
EXPLORER ... RSA_Encryption_Decryption.py X
C:\>Users\sonip\OneDrive\Desktop> RSA_Encryption.Decryption.py & generate_keys
1 import random
2
3 # Helper: Compute GCD
4 def gcd(a, b):
5     while b != 0:
6         a, b = b, a % b
7     return a
8
9 # Helper: Compute modular inverse using Extended Euclidean Algorithm
10 def modinv(a, m):
11     m0, x0, x1 = m, 0, 1
12     while a > 1:
13         q = a // m
14         m, a = a % m, m
15         x0, x1 = x1 - q * x0, x0
16     return x1 + m0 if x1 < 0 else x1
17
18 # Check for primality (naive, not suitable for large numbers)
19 def is_prime(n):
20     if n <= 1:
21         return False
22     for i in range(2, int(n**0.5)+1):
23         if n % i == 0:
24             return False
25     return True
26
27 # Generate RSA keys
28 def generate_keys(p, q):
29     if not (is_prime(p) and is_prime(q)):
30         raise ValueError('Both numbers must be prime.')
31     elif p == q:
32         raise ValueError('p and q cannot be equal')
33
34     n = p * q
35     phi = (p - 1) * (q - 1)
```

- Status Bar:** Line 31, Col 17, Spaces: 4, UTF-8, CRLF, Python 3.13.2.

The screenshot shows a Python code editor interface with the following details:

- File Explorer:** Shows a single file named "RSA_Encryption_Decryption.py" is open.
- Search Bar:** Contains the placeholder text "Search".
- Code Area:** Displays the following Python code:

```
def generate_keys(p, q):
    # Choose e
    e = 65537 # Common choice
    if gcd(e, phi) != 1:
        # Try finding an alternate e
        e = 3
        while gcd(e, phi) != 1:
            e += 2

    # Calculate d
    d = modinv(e, phi)

    return ((e, n), (d, n)) # public key, private key

# Encrypt message
def encrypt(public_key, plaintext):
    e, n = public_key
    cipher = [pow(ord(char), e, n) for char in plaintext]
    return cipher

# Decrypt message
def decrypt(private_key, ciphertext):
    d, n = private_key
    plain = [chr(pow(char, d, n)) for char in ciphertext]
    return ''.join(plain)

# Example usage
if __name__ == '__main__':
    p = 61
    q = 53
    public, private = generate_keys(p, q)

    message = "Hello RSA"
    print("Original message:", message)

    encrypted = encrypt(public, message)
    print("Encrypted message:", encrypted)

    decrypted = decrypt(private, encrypted)
    print("Decrypted message:", decrypted)
```

Status Bar: Shows "In 46 Col 23 Spaces: 4 UTR-8 CR LF {} Python 3.13.2".

Output:

The screenshot shows the Visual Studio Code interface. The left sidebar has 'EXPLORER' selected, showing a file tree with 'RSA_Encryption.Decryption.py'. The main editor area displays Python code for RSA encryption. The terminal at the bottom shows the execution of the script and its output.

```
File Edit Selection View Go Run Terminal Help ← → Search
EXPLORER RSA_Encryption.Decryption.py ...
OUTLINE C:\Users\sonip\OneDrive\Desktop> RSA_Encryption_Decryption.py > generate_keys
TIMELINE
MAVEN

28 def generate_keys(p, q):
37     # Choose e
38     e = 65537 # Common choice
39     if gcd(e, phi) != 1:
40         # Try finding an alternate e
41         e = 3
42         while gcd(e, phi) != 1:
43             e += 2
44
45     # Calculate d
46     d = modinv[e, phi]
47
48     return ((e, n), (d, n)) # public key, private key
49
50 # Encrypt message
51 def encrypt(public_key, plaintext):
52     e, n = public_key
53     cipher = [pow(ord(char), e, n) for char in plaintext]

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Code + v ... ^ x
PS C:\Users\sonip> python -u "c:\Users\sonip\OneDrive\Desktop\RSA_Encryption_Decryption.py"
Original message: Hello RSA
Encrypted message: [3800, 1313, 745, 745, 2185, 1992, 1859, 2680, 2790]
Decrypted message: Hello RSA
PS C:\Users\sonip>
```

Ln 46, Col 23 Spaces: 4 UTF-8 CR/LF {} Python 3.13.2

Conclusion:

Implementing RSA encryption and decryption in Python is a valuable exercise for understanding the fundamentals of public-key cryptography. It involves generating a pair of keys using prime numbers, encrypting data with the public key, and decrypting it with the private key. This process helps illustrate important mathematical concepts such as modular arithmetic, Euler's totient function, and modular inverses. Through this implementation, one gains insight into how secure communication can be established without the need for sharing a secret key in advance.