**FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING**

**Department of Computer Engineering**

**1. Course , Subject & Experiment Details**

|  |  |  |  |
| --- | --- | --- | --- |
| **Academic Year** | **2024-25** | **Estimated Time** | **03 - Hours** |
| **Course & Semester** | **T.E. (CMPN)- Sem VI** | **Subject Name & Code** | **CSS – CSC602** |
| **Chapter No.** | **02 – Mapped to CO2,CO3** | **Chapter Title** | **Basics of Cryptography** |

|  |  |
| --- | --- |
| **Practical No:** | **7** |
| **Title:** | Implementation and analysis of RSA cryptosystem and Digital signature scheme using RSA. |
| **Date of Performance:** | 03-04-2025 |
| **Date of Submission:** | 27-04-2025 |
| **Roll No:** | **9913** |
| **Name of the Student:** | **Mark Lopes** |

**Evaluation:**

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Rubric** | **Grade** |
| **1** | **On time submission Or completion (2)** |  |
| **2** | **Preparedness(2)** |  |
| **3** | **Skill (4)** |  |
| **4** | **Output (2)** |  |

**Signature of the Teacher:**

**Date:**

Lab Manual Prepared by : Prof. Monali Shetty

**Title:** Implementation and analysis of RSA cryptosystem and Digital signature scheme using RSA/ElGamal.

### Lab Objective:

This lab provides insight into:

* How the public-key algorithms work and understand the working of RSA.

**Reference** : “Cryptography and Network Security” B. A. Forouzan “Information Security Principles and Practice” Mark Stamp “Cryptography and Network Security” Atul Kahate

**Prerequisite :** Any programming language and Knowledge of Ciphering .

### Theory:

To overcome the problems faced in symmetric key algorithms, people have chosen Asymmetric Key algorithms for communication. Communication with Asymmetric algorithms will give us transmission of information without exchanging the key.

**Public-key cryptography** refers to a cryptographic system requiring two separate keys, one of which is secret and one of which is public. Public-key cryptography is widely used. It is an approach used by many cryptographic algorithms and cryptosystems. It underpins such Internet standards as Transport Layer Security(TLS), PGP, and GPG. RSA and Diffie–Hellman key exchange are the most widely used public key distribution systems, while the Digital Signature Algorithm is the most widely used digital signature system. Asymmetric algorithms which are mostly used are RSA cryptosystem and ElGamal Cryptosystem.

**The RSA algorithm** is the most commonly used encryption and authentication algorithm and is included as part of the Web browsers from Microsoft and Netscape.RSA is an algorithm for public key cryptography that is based on the presumed difficulty of factoring large integers, the factoring problem.The RSA algorithm involves threesteps: key generation, encryption and decryption.

**ElGamal System** is a public-key cryptosystem based on the discrete logarithm problem. It consists of both encryption and Signature algorithms. ElGamal encryption is used in the free GNU Privacy Guard software, recent versions of PGP, and other cryptosystems. ElGamal encryption consists of three components: the key generator, the encryptionalgorithm, and the decryption algorithm.

**ALGORITHM RSA**

Lab Manual Prepared by : Prof. Monali Shetty

**Example of RSA**

### >> Generating Public Key :

* Select two prime no's. Suppose **P = 53 and Q = 59**. Now First part of the Public key : **n = P\*Q = 3127**.
* We also need a small exponent say **e** :But e Must be

An integer.

Not be a factor of n.

**1 < e < Φ(n)** \*Φ(n) is discussed below+, Let us now consider it to be equal to 3.

* Our Public Key is made of n and e

### >> Generating Private Key:

Lab Manual Prepared by : Prof. Monali Shetty



We need to calculate Φ(n) : Such that **Φ(n) = (P-1)(Q 1)** so, Φ(n) = 3016

Now calculate Private Key, **d** :

**d = (k\*Φ(n) + 1) / e** for some integer

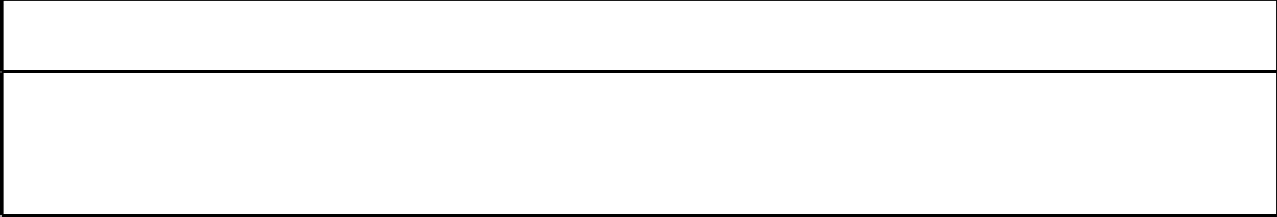
kFor k = 2, value of d is 201

## Now we are ready with our – Public Key ( n = 3127 and e = 3) and Private Key(d = 2011)Now we will encrypt **“HI”** :

* Convert letters to numbers : H = 8 and I = 9
* Thus **Encrypted Data c = 89e mod n**.

Thus our Encrypted Data comes out to be 1394

|  |  |
| --- | --- |
| Now we will decrypt **1349** : | |
|  | * **Decrypted Data = cd mod n**.   Thus our Encrypted Data comes out to be 89 |
| **8 = H and I = 9 i.e. "HI".** | |



**Conclusion:**

The program was tested for different sets of inputs.

Program is working SATISFACTORY NOT SATISFACTORY ( Tick

appropriate outcome)



**Post Lab Assignment:**

Test above an experiment to estimate the amount of time to

1. Generate key pair (RSA )
2. Encrypt n bit message (RSA)
3. Decrypt n bit message (RSA)

As function of key size, experiment with different n-bit messages. Summarize your Conclusion.

Lab Manual Prepared by : Prof. Monali Shetty Lab Manual Prepared by : Prof. Monali Shetty

# Rsa\_Server.py

import socket import hashlib

def power(base, expo, m): res = 1

base = base % m while expo > 0:

if expo & 1:

res = (res \* base) % m base = (base \* base) % m expo //= 2

return res

def modInverse(e, phi): for d in range(2, phi):

if (e \* d) % phi == 1: return d

return -1

def gcd(a, b): while b:

a, b = b, a % b return a

def generateKeys(p, q): n = p \* q

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

e = next(i for i in range(2, phi) if gcd(i, phi) == 1) d = modInverse(e, phi)

return e, d, n

def encrypt(m, e, n): return power(m, e, n)

def decrypt(c, d, n): return power(c, d, n)

def text\_to\_number(text):

return ' '.join(str(ord(c)) for c in text)

def number\_to\_text(number\_str):

return ''.join(chr(int(n)) for n in number\_str.split())

def encrypt\_text(text, e, n): num\_str = text\_to\_number(text)

encrypted\_numbers = [str(encrypt(int(num), e, n)) for num in num\_str.split()]

return ' '.join(encrypted\_numbers)

p\_server, q\_server = 7919, 1009

e\_server, d\_server, n\_server = generateKeys(p\_server, q\_server)

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) server\_socket.bind(('0.0.0.0', 12345))

server\_socket.listen(1)

conn, addr = server\_socket.accept()

conn.send(f"{e\_server},{n\_server}".encode()) client\_key\_data = conn.recv(4096).decode()

e\_client, n\_client = map(int, client\_key\_data.split(','))

message = "This is a secure message from the server." hashed\_msg = hashlib.md5(message.encode()).hexdigest()

encrypted\_message = encrypt\_text(message, e\_client, n\_client) print("Encrypted message: ", encrypted\_message)

signature = encrypt\_text(hashed\_msg, d\_server, n\_server) print("Digital Signature: ", signature)

conn.send(f"{encrypted\_message}||{signature}".encode()) conn.close()

server\_socket.close()

# Rsa\_Client.py:

import socket import hashlib

def power(base, expo, m): res = 1

base = base % m while expo > 0:

if expo & 1:

res = (res \* base) % m base = (base \* base) % m expo //= 2

return res

def modInverse(e, phi): for d in range(2, phi):

if (e \* d) % phi == 1: return d

return -1

def gcd(a, b): while b:

a, b = b, a % b return a

def generateKeys(p, q): n = p \* q

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

e = next(i for i in range(2, phi) if gcd(i, phi) == 1) d = modInverse(e, phi)

return e, d, n

def encrypt(m, e, n): return power(m, e, n)

def decrypt(c, d, n): return power(c, d, n)

def text\_to\_number(text):

return ' '.join(str(ord(c)) for c in text)

def number\_to\_text(number\_str):

return ''.join(chr(int(n)) for n in number\_str.split())

def encrypt\_text(text, e, n): num\_str = text\_to\_number(text)

encrypted\_numbers = [str(encrypt(int(num), e, n)) for num in num\_str.split()] return ' '.join(encrypted\_numbers)

def decrypt\_text(encrypted\_str, d, n): encrypted\_numbers = encrypted\_str.split()

decrypted\_numbers = [str(decrypt(int(num), d, n)) for num in encrypted\_numbers]

return number\_to\_text(' '.join(decrypted\_numbers))

p\_client, q\_client = 7877, 1013

e\_client, d\_client, n\_client = generateKeys(p\_client, q\_client)

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) client\_socket.connect(('127.0.0.1', 12345))

server\_key\_data = client\_socket.recv(4096).decode() e\_server, n\_server = map(int, server\_key\_data.split(','))

client\_socket.send(f"{e\_client},{n\_client}".encode())

data = client\_socket.recv(8192).decode() encrypted\_message, signature = data.split('||') print("Received data: ", data)

message = decrypt\_text(encrypted\_message, d\_client, n\_client) print("Decrypted message: ", message)

# message += "!"

hashed\_msg = hashlib.md5(message.encode()).hexdigest() print("Message hashed value: ", hashed\_msg)

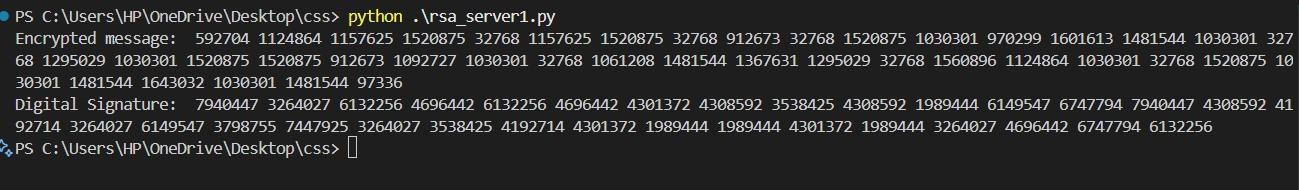
decrypted\_signature = decrypt\_text(signature, e\_server, n\_server) print("Digital Signature hashed: ", decrypted\_signature)

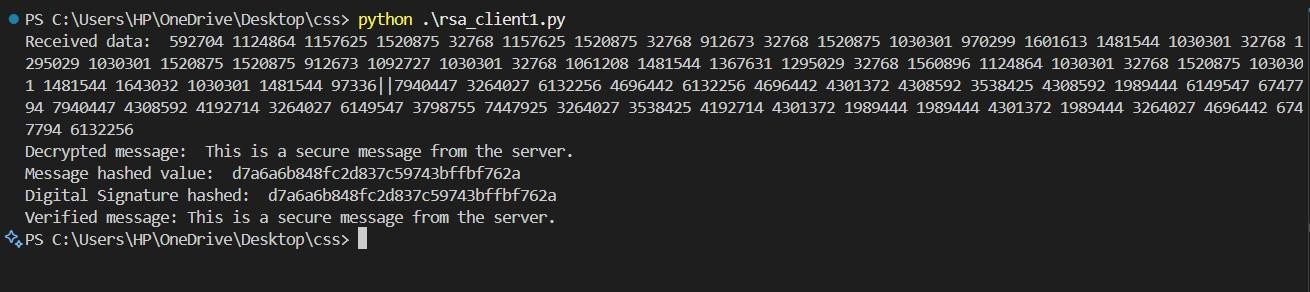
if hashed\_msg == decrypted\_signature: print(f"Verified message: {message}")

else:

print("Signature verification failed.") client\_socket.close()

OUTPUT:

Server.py

Client.py