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

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 = 89^e mod n**.
  Thus our Encrypted Data comes out to be 1394

Now we will decrypt **1349** :

- **Decrypted Data = c^d 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
i) Generate key pair (RSA)
ii) Encrypt n bit message (RSA)
iii) 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

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

```
PS C:\Users\HP\OneDrive\Desktop\css> python .\rsa_server1.py
Encrypted message:  592704 1124864 1157625 1520875 32768 1157625 1520875 32768 912673 32768 1520875 1030301 970299 1601613 1481544 1030301 327
68 1295029 1030301 1520875 1520875 912673 1092727 1030301 32768 1061208 1481544 1367631 1295029 32768 1560896 1124864 1030301 32768 1520875 10
30301 1481544 1643032 1030301 1481544 97336
Digital Signature:  7940447 3264027 6132256 4696442 6132256 4696442 4301372 4308592 3538425 4308592 1989444 6149547 6747794 7940447 4308592 41
92714 3264027 6149547 3798755 7447925 3264027 3538425 4192714 4301372 1989444 1989444 4301372 1989444 3264027 4696442 6747794 6132256
PS C:\Users\HP\OneDrive\Desktop\css>
```

Client.py

```
PS C:\Users\HP\OneDrive\Desktop\css> python .\rsa_client1.py
Received data:  592704 1124864 1157625 1520875 32768 1157625 1520875 32768 912673 32768 1520875 1030301 970299 1601613 1481544 1030301 32768 1
295029 1030301 1520875 1520875 912673 1092727 1030301 32768 1061208 1481544 1367631 1295029 32768 1560896 1124864 1030301 32768 1520875 103030
1 1481544 1643032 1030301 1481544 97336||7940447 3264027 6132256 4696442 6132256 4696442 4301372 4308592 3538425 4308592 1989444 6149547 67477
94 7940447 4308592 4192714 3264027 6149547 3798755 7447925 3264027 3538425 4192714 4301372 1989444 1989444 4301372 1989444 3264027 4696442 674
7794 6132256
Decrypted message:  This is a secure message from the server.
Message hashed value:  d7a6a6b848fc2d837c59743bffbf762a
Digital Signature hashed:  d7a6a6b848fc2d837c59743bffbf762a
Verified message: This is a secure message from the server.
PS C:\Users\HP\OneDrive\Desktop\css>
```