**Department of Computer Engineering**

**Name of the Student: Parth Das Roll Number: C-111**

**SAP ID: 60004220185 Class: Comps Division: C-2 Batch: 2**

**Subject: Information Security**

**DATE OF PERFORMANCE: 05/04/2025 DATE OF SUBMISION: 05/04/2025**

**EXPERIMENT NO: 8**

**AIM:** Implement Digital Signatures using RSA and Elgamal Digital Signature Scheme.

**Theory:**

**SOURCE CODE WITH OUTPUTS:**

1. **Digital Signatures using RSA**

import hashlib

# ---------- Step 1: SHA-1 Hash ----------

def hash\_with\_sha1(message):

encoded\_msg = message.encode()

hash\_object = hashlib.sha1(encoded\_msg)

hex\_dig = hash\_object.hexdigest()

return hex\_dig

# ---------- Step 2: RSA Keys (Small, for demo only) ----------

n = 3233

e = 17

d = 2753

# ---------- Step 3: Sign Function ----------

def sign\_from\_hash(hex\_hash):

hash\_int = int(hex\_hash, 16) % n # MODULO to fit in range of n

signature = pow(hash\_int, d, n)

return signature, hash\_int # returning both for verification

# ---------- Step 4: Verify Function ----------

def verify\_from\_hash(hash\_int, signature):

decrypted = pow(signature, e, n)

return hash\_int == decrypted

# ---------- Step 5: Run ----------

message = "InformationSecurity123"

hex\_hash = hash\_with\_sha1(message)

signature, reduced\_hash = sign\_from\_hash(hex\_hash)

print("Original Message:", message)

print("SHA-1 Hash:", hex\_hash)

print("Reduced Hash (mod n):", reduced\_hash)

print("RSA Signature:", signature)

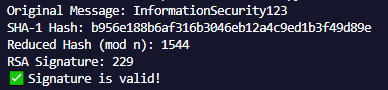
if verify\_from\_hash(reduced\_hash, signature):

print("✅ Signature is valid!")

else:

print("❌ Signature is invalid.")

**Output:-**



**SOURCE CODE WITH OUTPUTS:**

1. **Elgamal Digital Signatures Scheme**

import random

import hashlib

def modinv(a, m):

m0, x0, x1 = m, 0, 1

if m == 1:

return 0

while a > 1:

q = a // m

a, m = m, a % m

x0, x1 = x1 - q \* x0, x0

return x1 % m0

def hash\_message(message):

hash\_object = hashlib.sha1(message.encode())

hex\_dig = hash\_object.hexdigest()

return int(hex\_dig, 16)

# Key Generation

def generate\_keys(p, g):

x = random.randint(1, p-2) # Private key

y = pow(g, x, p) # Public key

return x, y

def sign(message, p, g, x):

H = hash\_message(message)

while True:

k = random.randint(1, p-2)

if gcd(k, p-1) == 1:

break

r = pow(g, k, p)

k\_inv = modinv(k, p-1)

s = (k\_inv \* (H - x \* r)) % (p - 1)

return r, s

def verify(message, r, s, p, g, y):

if not (0 < r < p):

return False

H = hash\_message(message)

v1 = (pow(y, r, p) \* pow(r, s, p)) % p

v2 = pow(g, H, p)

return v1 == v2

def gcd(a, b):

while b:

a, b = b, a % b

return a

p = 467 # Prime number

g = 2 # Generator

message = "InformationSecurity123"

x, y = generate\_keys(p, g)

print("Private key x:", x)

print("Public key y:", y)

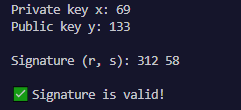
r, s = sign(message, p, g, x)

print("\nSignature (r, s):", r, s)

is\_valid = verify(message, r, s, p, g, y)

print("\n✅ Signature is valid!" if is\_valid else "❌ Signature is invalid.")

**Output:-**



**CONCLUSION:**

Digital signatures using RSA and ElGamal schemes play a crucial role in ensuring authenticity, integrity, and non-repudiation of messages. RSA digital signatures are deterministic and rely on the hardness of factoring large integers, offering both encryption and signing in a unified system. In contrast, ElGamal digital signatures are probabilistic, using random values for each signature, which makes them more resistant to certain types of attacks.