**ASSIGNMENT NO : 10**

**NAME : PARAS WANE**

**PRN: 21510041**

**BATCH : B2**

1. **Q - Implement the SIGNATURE SCHEME – Digital Signature Standard.**

import hashlib

import random

# Helper functions

def mod\_inverse(k, q):

    """ Compute modular inverse of k mod q using Extended Euclidean Algorithm. """

    r0, r1 = q, k

    s0, s1 = 1, 0

    while r1 > 0:

        q = r0 // r1

        r0, r1 = r1, r0 - q \* r1

        s0, s1 = s1, s0 - q \* s1

    if r0 != 1:

        raise ValueError("Inverse doesn't exist")

    return s0 % q

# Key generation

def generate\_keys(p, q):

    """ Generate public and private keys. """

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

    y = pow(2, x, p)               # Public key

    return x, y

# Signing

def sign\_message(p, q, g, x, message):

    """ Sign a message using the private key x. """

    H = int(hashlib.sha256(message.encode()).hexdigest(), 16)  # Hash the message

    k = random.randint(1, q - 1)  # Random k

    r = pow(g, k, p) % q          # r = (g^k mod p) mod q

    k\_inv = mod\_inverse(k, q)      # k^(-1) mod q

    s = (k\_inv \* (H + x \* r)) % q  # s = k^(-1) \* (H + x \* r) mod q

    return r, s

# Verification

def verify\_signature(p, q, g, y, message, signature):

    """ Verify the signature of the message. """

    r, s = signature

    H = int(hashlib.sha256(message.encode()).hexdigest(), 16)  # Hash the message

    if not (0 < r < q) or not (0 < s < q):

        return False

    w = mod\_inverse(s, q)

    u1 = (H \* w) % q

    u2 = (r \* w) % q

    v = (pow(g, u1, p) \* pow(y, u2, p)) % p % q  # v = (g^u1 \* y^u2 mod p) mod q

    return v == r

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

    # Parameters (for demonstration purposes, in practice, use larger values)

    p = 23  # A prime number

    q = 11  # A prime divisor of p-1

    g = 2   # A generator

    # Generate keys

    private\_key, public\_key = generate\_keys(p, q)

    # Sign a message

    message = "Hello, this is a test message!"

    signature = sign\_message(p, q, g, private\_key, message)

    # Verify the signature

    is\_valid = verify\_signature(p, q, g, public\_key, message, signature)

    print(f"Message: {message}")

    print(f"Signature: {signature}")

    print(f"Is the signature valid? {is\_valid}")

OUTPUT :

