Name: **Mareena Fernandes**
Roll No.: **8669**
Class: **TE IT**
Batch: **B**

EXPERIMENT NO: 2

RSA Sign:

```
def is_prime(num):
    for i in range(2, num):
        if num % i == 0:
            return False
    return True


def gcd(n1, n2):
    if n2 == 0:
        return n1
    else:
        return gcd(n2, n1 % n2)


while True:
    key_p = int(input("Enter a large prime number: "))
    if not is_prime(key_p):
        print("Not a prime number!")
        continue

    key_q = int(input("Enter another large prime number: "))
    if not is_prime(key_q):
        print("Not a prime number!")
        continue

    key_phi = (key_p - 1) * (key_q - 1)
    key_n = key_p * key_q
    if key_n < 100:
        print("Invalid pair of primes. Try larger primes")

    key_e = int(input("Enter a number less than and coprime to " + str(key_phi) + ": "))
```

```python
key_d = None
valid = True

# for i in range(2, key_e + 1):
#     if key_e % i == 0 and key_phi % i == 0:
#         valid = False

if not gcd(key_phi, key_e) == 1:
    print("Invalid coprime!")
    continue

for i in range(1, key_n):
    if (i * key_e) % key_phi == 1:
        key_d = i

pub_key = [key_n, key_e]

message = input("Enter the message: ").upper()
enc_msg = []
for letter in message:
    letter = ord(letter)
    print(letter)
    enc_msg.append((letter ** key_d) % key_n)

signature = [message, enc_msg, pub_key]
print("Signed hash: ", enc_msg)

og_msg = []
for num in enc_msg:
    og_msg.append(chr((num ** key_e) % key_n))
print("Message retrieved from hash: ", og_msg)
```

Elgamal Sign:

```
def is_prime(num):
    for i in range(2, num):
        if num % i == 0:
            return False
    return True


def gcd(n1, n2):
    if n2 == 0:
        return n1
    else:
        return gcd(n2, n1 % n2)


def power_mod(base, power, mod):
    res = 1
    while power > 0:
        if power % 2 == 1:
            res = (res * base) % mod
        power = power // 2
        base = (base * base) % mod
    return res


def prime_factors_of(phi, prime_factors):
    while phi % 2 == 0:
        prime_factors.add(2)
        phi = phi // 2

    i = 3
    while i * i <= phi:
        while phi % i == 0:
            prime_factors.add(i)
            phi = phi // i
        i += 2

    if phi > 2:
        prime_factors.add(phi)
```

```python
def prim_root_of(p):
    prime_factors = set()
    phi = p - 1
    prime_factors_of(phi, prime_factors)

    powers = set()
    for factor in prime_factors:
        powers.add(phi // factor)

    for base in range(2, phi + 1):
        found = True
        for power in powers:
            if power_mod(base, power, p) == 1:
                found = False
                break
        if found:
            return base
    return False


while True:
    p = int(input("Enter a large prime: "))
    if not is_prime(p):
        print("Invalid prime")
        continue
    phi = p - 1

    e1 = prim_root_of(p)
    if not e1:
        print("No primitive root found. Pick another large prime")
        continue

    d = int(input("Enter a secret nonce less than " + str(phi - 1) + ": "))
    if d >= p - 2 or d < 1:
        print("Invalid secret nonce")
        continue
```

```
e2 = 0
if (d > 10) or (e1 > 10):
    e2 = power_mod(e1, d, p)
else:
    e2 = (e1 ** d) % p

k = int(input("Enter a random integer coprime to " + str(phi) + ": "))
if gcd(k, p - 1) != 1:
    print("Invalid coprime")
    continue

k_inv = False
for i in range(1, p):
    if (i * k) % phi == 1:
        k_inv = i
if not k_inv:
    print("Invalid coprime. No inverse found")
    continue

msg = int(input("Enter message digest: "))

ciph_1 = power_mod(e1, k, p)

ciph_2 = (k_inv * (msg - d * ciph_1)) % phi

public_key = [e1, e2, p]
signature = (ciph_1, ciph_2, msg, public_key)
print("Signature cipher pairs: (", ciph_1, ",", ciph_2, ")")
print("Public Key: ", public_key)

# Verification
ver_1_half_1 = power_mod(signature[3][1], signature[0], signature[3][2])
ver_1_half_2 = power_mod(signature[0], signature[1], signature[3][2])
ver_1 = (ver_1_half_1 * ver_1_half_2) % signature[3][2]

ver_2 = power_mod(signature[3][0], signature[2], signature[3][2])

print("Verification code 1: ", ver_1)
```

```
print("Verification code 2: ", ver_2)
if ver_1 == ver_2:
    print("Verification successful since both codes match")
else:
    print("Invalid signature. Verification code mismatch")
```

Post labs:

1. Explain direct and arbitrated digital signature.

Ans:
Digital signature is cryptographic method for ensuring sender authentication and integrity. Direct digital signature is a direct communication of digitally signed message from sender to receiver. In direct digital signature, the sender and receiver trust each other and are in possession of each other’s public keys. The message in encrypted using sender’s private key for authentication as digital signature. It can be then encrypted using receiver’s public key to maintain confidentiality.

Arbitrated digital signature includes a trusted third party or arbiter. All messages pass through this trusted arbiter. The arbiter performs certain checks on the signed message to verify the signature. When satisfied with its authenticity, arbiter adds a timestamp to verify receipt and sends to receiver. It used private key method.