CRYPTOGRAPHY- LAB-13

NAME: Shlok Kamath

REG NO: 21BAI1844

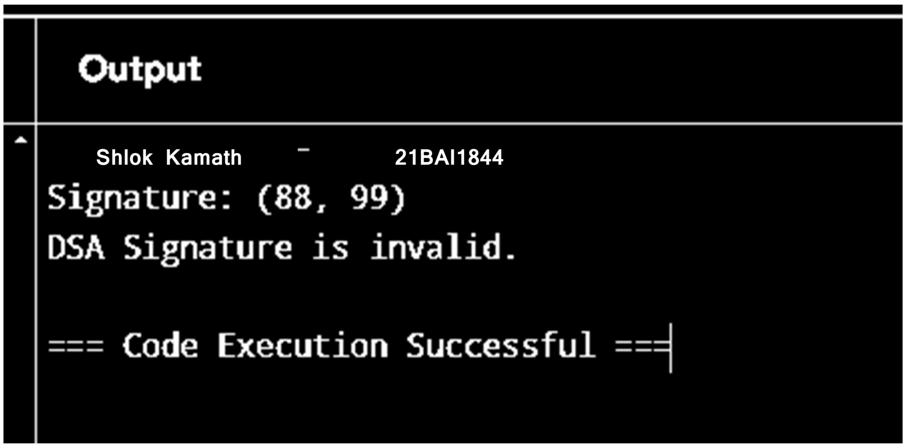
1) **DSA**

CODE:

```
import random
from hashlib import sha256

# Helper function to compute modular inverse (Extended Euclidean Algorithm)
def mod_inverse(a, m):
    m0, x0, x1 = m, 0, 1
    if m == 1:
        return 0
    while a > 1:
        q = a // m
        m, a = a % m, m
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1
p = 467
q = 233
g = 2
def dsa_keygen(p, q, g):
    x = random.randint(1, q - 1)
    y = pow(g, x, p)
    return (p, q, g, y), x
def dsa_sign(message, p, q, g, x):
    while True:
        k = random.randint(1, q - 1)  # Random per-signature key
        if mod_inverse(k, q) is not None:  # Ensure k has an inverse
            break
```

```
    r = pow(g, k, p) % q
    k_inv = mod_inverse(k, q)
    message_hash = int.from_bytes(sha256(message).digest(), byteorder='big') % q
    s = (k_inv * (message_hash + x * r)) % q
    return r, s

# DSA Signature Verification
def dsa_verify(message, signature, p, q, g, y):
    r, s = signature
    if not (0 < r < q and 0 < s < q):
        return False
    w = mod_inverse(s, q)
    message_hash = int.from_bytes(sha256(message).digest(), byteorder='big') % q
    u1 = (message_hash * w) % q
    u2 = (r * w) % q
    v = ((pow(g, u1, p) * pow(y, u2, p)) % p) % q
    return v == r

# Generate keys
public_key, private_key = dsa_keygen(p, q, g)

# Sign the message
message = b"This is a secret message"
signature = dsa_sign(message, p, q, g, private_key)
print(f"Signature: {signature}")

# Verify the signature
is_valid = dsa_verify(message, signature, p, q, g, public_key[3])
print("DSA Signature is valid." if is_valid else "DSA Signature is invalid.")
```

Output:



2) **ELGAMAL**
CODE:

```
import random
from hashlib import sha256
from math import gcd

# ElGamal parameters
def mod_inverse(a, m):
    m0, x0, x1 = m, 0, 1
    if m == 1:
        return 0
    while a > 1:
        q = a // m
        m, a = a % m, m
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1

# ElGamal Key Generation
def elgamal_keygen(p, g):
    x = random.randint(1, p - 2)  # Private key
    y = pow(g, x, p)  # Public key
    return (p, g, y), x

# ElGamal Signature
def elgamal_sign(message, p, g, x):
    while True:
        k = random.randint(1, p - 2)
        if gcd(k, p - 1) == 1:
            break
    r = pow(g, k, p)
    k_inv = mod_inverse(k, p - 1)
    m_hash = int.from_bytes(sha256(message).digest(), byteorder='big')
    s = (k_inv * (m_hash - x * r)) % (p - 1)
    return r, s
```

```
# ElGamal Signature Verification
def elgamal_verify(message, signature, p, g, y):
    r, s = signature
    if not (0 < r < p):
        return False
    m_hash = int.from_bytes(sha256(message).digest(), byteorder='big')
    v1 = pow(g, m_hash, p)
    v2 = (pow(y, r, p) * pow(r, s, p)) % p
    return v1 == v2

# Parameters (for real applications, use large prime numbers)
p = 467  # Large prime number
g = 2  # Generator

# Key generation
public_key, private_key = elgamal_keygen(p, g)
message = b"ElGamal signature test"

# Signing
signature = elgamal_sign(message, p, g, private_key)
print(f"Signature: {signature}")

# Verification
is_valid = elgamal_verify(message, signature, p, g, public_key[2])
print("ElGamal Signature is valid." if is_valid else "ElGamal Signature is
invalid.")
```

OUTPUT: