Christ University

Name: Joel Joseph Motha **Regno:** 2448521

Course: Quantum Computing **Component:** CIA 3 - B

Question 1

1. Explain the resource advantage of superdense coding?

The superdense coding protocol provides a quantum advantage by doubling the classical information capacity of a qubit. By utilizing a pre-shared entangled resource (a Bell pair), Alice can transmit two classical bits of information to Bob by physically sending only one qubit. In classical physics, sending a single two-state particle could strictly convey only one bit of information.

2. How many qubits are physically transmitted from Alice to Bob?

One qubit. Alice operates on her half of the entangled pair and sends only that single specific qubit through the channel to Bob.

3. How many classical bits are successfully communicated upon measurement?

Two classical bits. Upon performing a Bell measurement (CNOT followed by Hadamard) on the pair, Bob successfully reconstructs the specific two-bit message ($00, 01, 10,$ or 11) Alice intended to send.

Implementation:

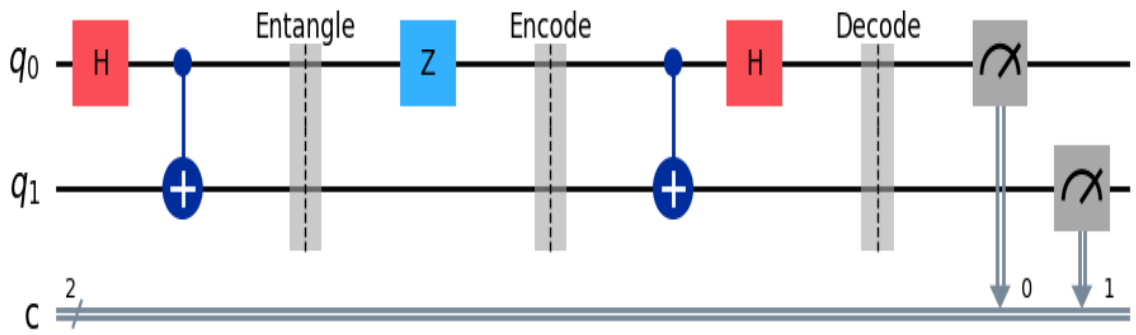
```
from qiskit import QuantumCircuit, transpile

from qiskit_aer import AerSimulator


def superdense_coding_protocol(message):

    # 1. Setup Circuit

    qc = QuantumCircuit(2, 2)


    # 2. Bell Pair

    qc.h(0)
```

```
qc.cx(0, 1)

# 3. Alice's Encoding
if message == '00':
    pass
elif message == '01':
    qc.z(0)
elif message == '10':
    qc.x(0)
elif message == '11':
    qc.z(0)
    qc.x(0)

# 4. Bob's Decoding
qc.cx(0, 1)
qc.h(0)

# 5. Measure
qc.measure([0, 1], [0, 1])

# Execute
backend = AerSimulator()
qc_compiled = transpile(qc, backend)
job = backend.run(qc_compiled, shots=1)
result = job.result().get_counts()
```

```
    return list(result.keys())[0]


inputs = ["00", "01", "10", "11"]


print(f"{'Input':<10} | {'Received':<10} | {'Status'}")
print("-" * 35)


for msg in inputs:
    received = superdense_coding_protocol(msg)
    status = "✅ PASS" if msg == received else "❌ FAIL"
    print(f"{msg:<10} | {received:<10} | {status}")
```

Output:

```
Input      | Received   | Status
-----------------------------------
00         | 00         | ✅ PASS
01         | 01         | ✅ PASS
10         | 10         | ✅ PASS
11         | 11         | ✅ PASS
```



Proof & Analysis:

```
   SUPERDENSE CODING: COMPUTATIONAL PROOF (Message '11')
STEP 1: INITIAL STATE
---------------------
Math: We start with two independent qubits in state |0>.
Current State: [1.+0.j,0.+0.j,0.+0.j,0.+0.j]

STEP 2: CREATING ENTANGLEMENT (The Bell Pair)
-----------------------------------------
Action: Alice applies H, then CNOT(0,1).
Math:   This creates the superposition (|00> + |11>) / sqrt(2).
Current State: [0.70710678+0.j,0.        +0.j,0.        +0.j,0.70710678+0.j]

STEP 3a: ALICE APPLIES Z-GATE (Phase Flip)
------------------------------------------
Action: Alice applies Z to qubit 0.
Math:   The sign of the |1> term flips. State becomes (|00> - |11>) / sqrt(2).
Current State: [ 0.70710678+0.j, 0.        +0.j, 0.        +0.j,-0.70710678+0.j]

STEP 3b: ALICE APPLIES X-GATE (Bit Flip)
----------------------------------------
Action: Alice applies X to qubit 0.
Math:   0 becomes 1, and 1 becomes 0.
        |00> -> |10>  and  |11> -> |01>
        Resulting State: (|10> - |01>) / sqrt(2)
Current State: [ 0.        +0.j, 0.70710678+0.j,-0.70710678+0.j, 0.        +0.j]
...
============================================================
CONCLUSION: The final state is strictly |11>.
            Measurement will yield '11' with 100% probability.
============================================================
```

Question 2:

- Diffie–Hellman lets two parties create a shared secret key over an insecure channel using modular arithmetic.
- Public values: a large prime p and generator g
- Private values: Alice picks a, Bob picks b

They exchange:

$$A = g^a \pmod{p}$$
$$B = g^b \pmod{p}$$

Both compute the same shared key:

$$S = g^{ab} \pmod{p}$$

- Security comes from the difficulty of the Discrete Logarithm Problem. Standard DH has no authentication, making it vulnerable to Man-in-the-Middle (MitM) attacks where an attacker intercepts the exchanged values and impersonates both parties.

Implementation:

```
def diffie_hellman_simulation():
    print("--- Diffie-Hellman Key Exchange Simulation ---\n")

    # 1. Agree on public parameters
    p = 23  # Prime number
    g = 5   # Generator
    print(f"Public Parameters: Prime (p)={p}, Generator (g)={g}")

    # 2. Assign Private Keys (Secret)
    a = 6   # Alice's private key
    b = 15  # Bob's private key
```

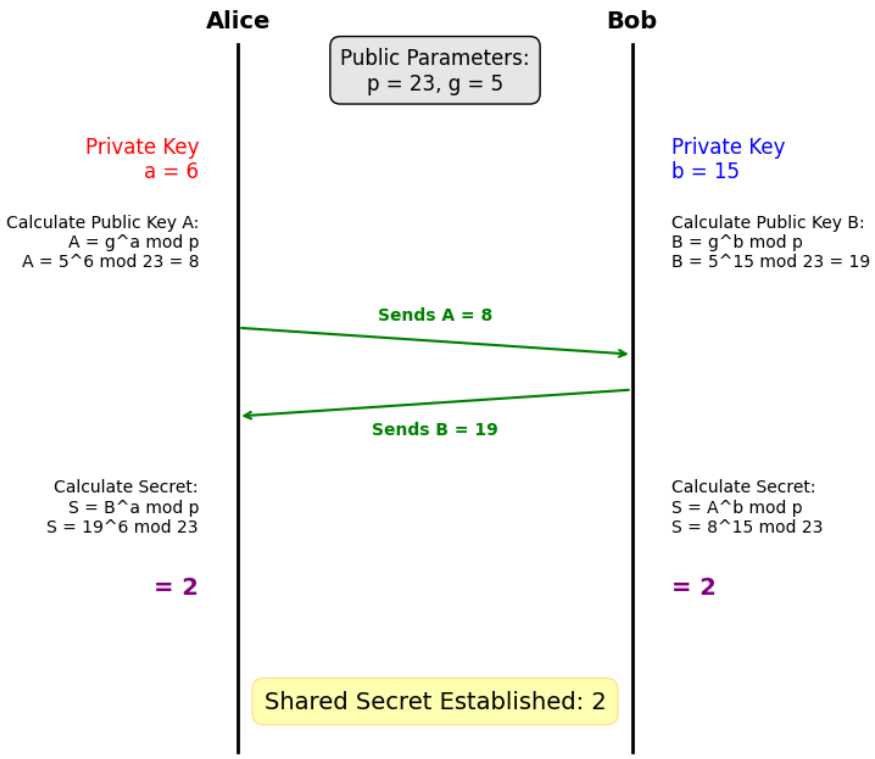
```
print(f"Alice's Private Key (a): {a}")
print(f"Bob's Private Key (b):   {b}")

# 3. Calculate Public Keys
# Alice computes A = g^a mod p
alice_public = (g ** a) % p

# Bob computes B = g^b mod p
bob_public = (g ** b) % p

print(f"\nCalculated Public Keys:")
print(f"Alice's Public Key (A) = {g}^{a} mod {p} = {alice_public}")
print(f"Bob's Public Key (B)   = {g}^{b} mod {p} = {bob_public}")

# 4. Compute Shared Secret
# Alice computes Sa = B^a mod p (using Bob's public key)
shared_secret_alice = (bob_public ** a) % p

# Bob computes Sb = A^b mod p (using Alice's public key)
shared_secret_bob = (alice_public ** b) % p

print(f"\nComputing Shared Secrets:")
print(f"Alice calculates: {bob_public}^{a} mod {p} = {shared_secret_alice}")
print(f"Bob calculates:   {alice_public}^{b} mod {p} = {shared_secret_bob}")

# 5. Programmatic Assertion
```

```
    try:
        assert shared_secret_alice == shared_secret_bob
        print(f"\n✅ SUCCESS: Shared secrets match! Key = {shared_secret_alice}")
    except AssertionError:
        print(f"\n❌ ERROR: Shared secrets do not match.")


if __name__ == "__main__":
    diffie_hellman_simulation()
```

Output:

```
--- Diffie-Hellman Key Exchange Simulation ---

Public Parameters: Prime (p)=23, Generator (g)=5
Alice's Private Key (a): 6
Bob's Private Key (b):   15

Calculated Public Keys:
Alice's Public Key (A) = 5^6 mod 23 = 8
Bob's Public Key (B)   = 5^15 mod 23 = 19

Computing Shared Secrets:
Alice calculates: 19^6 mod 23 = 2
Bob calculates:   8^15 mod 23 = 2

✅ SUCCESS: Shared secrets match! Key = 2
```

Diffie-Hellman Key Exchange Simulation



Proof & Analysis:

```
--- COMPUTATIONAL PROOF: Diffie-Hellman (p=23, g=5) ---

STEP 1: Alice's Public Key (A)
   Formula: 5^6 mod 23
   Intermediate: 5^6 = 15625
   Modulo:       15625 / 23 = 679 remainder 8
   Result (A):   8

STEP 2: Bob's Public Key (B)
   Formula: 5^15 mod 23
   Intermediate: 5^15 = 30517578125
   Modulo:       30517578125 / 23 = 1326851222 remainder 19
   Result (B):   19

STEP 3: Verify Shared Secret (S)
   Alice computes: 19^6 mod 23
   Intermediate:   19^6 = 47045881
   Modulo:         47045881 % 23 = 2
   Alice's Secret: 2

   Bob computes:   8^15 mod 23
   Intermediate:   8^15 = 35184372088832
   Modulo:         35184372088832 % 23 = 2
   Bob's Secret:   2

CONCLUSION: Mathematical verification SUCCESS. Both derived 2.
```