PROJECT DEMONSTRATION

This is the implementation, working, and outcome of the Quantum Cryptography key distribution system using the BB84 protocol with the support of IBM Qiskit Composer.

The principles of quantum key distribution, especially its contribution toward secure communication. The role of quantum mechanics in generation of private keys that are distributed to all the users involved in the communication, such as through superposition and measurement collapse.

- **Main Objective:** Provide key establishment between Alice, sender, and Bob, receiver, with the principles of quantum. Also provide simulation of all the intermeditiary steps using proper tools

- **Tools and Environment**

  - **Software Used :** AerSimulator and IBM Qiskit Composer.
  - **Language:** Python, primarily for simulation and OpenQASM, which represents low-level circuit design.
  - **Framework:** Quantum Circuit Design using Qubits and Gates Encoding, Transmission, and Measurement of Quantum States.

IBM Qiskit with Aer simulation: (Visualization)

Step 1: Initialization with IBM Qiskit – Using Python

Starting with the IBM Qiskit interface then go ahead with initialization of quantum registers, or qubits, and classical registers used for storing measurement results. Configuration of AerSimulator as per the objective to simulate quantum circuits.

Step 2: Alice's Encoding

- Show the random generation of Alice's bits and bases:
- Express randomness of selecting 0 or 1, computational or diagonal basis.
- Prove that Alice encodes the qubits:
  - Apply the X gate on bit 1.
  - Apply the H gate for diagonal basis.

Example:

Qubit 0:
Bit: 1, Basis Diagonal → Apply X then H.
Qubit 1:
Bit: 0, Basis: Computation → No gate applied.

Step 3: Bob Measures

- Show the random generation of Bob's bases:
- Bob measures the qubits.
- If the basis is diagonal, apply the H gate before measurement.
- Measure the qubits and store their results in bob_bits.

Example:

Qubit 0:
Basis: Diagonal → Apply H, then measure.
Qubit 1:
Basis: Computational → Direct measurement.

Step 4: Classical post-processing

- Broadcast Alice's and Bob's bases over a simulated public channel.
- Compare the bases:
- Keep bits if bases match.
- Discard those with different bases.

Example:

Alice's Bits: [1, 0, 1, 1, 0]
Alice's Bases: [Diagonal, Computational, Computational, Diagonal, Diagonal]
Bob's Bases: Diagonal, Diagonal, Computational, Diagonal, Computational
Held Key: [1, 1] (Matched indices: 0 and 3).

- **Simulation Execution**

Run the Python code in the AerSimulator. Output is Alice's bits, Alice's bases, Bob's bases, Bob's bits, the matched indices and the final key.

6.1: IBM Composer Circuit: (Visualization)

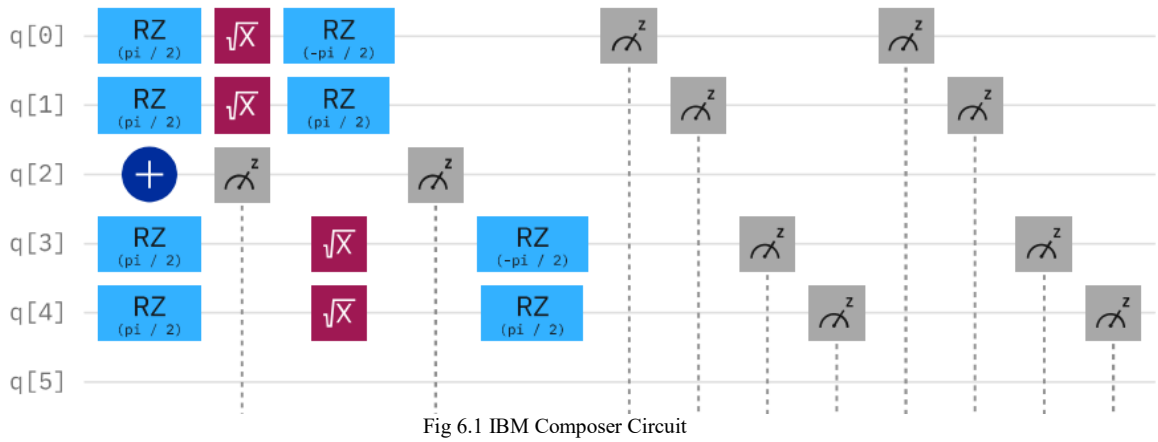

Fig 6.1 IBM Composer Circuit

- **Walkthrough of the Quantum part of Simulation (Step-by-Step Process)**

Step 1: Initialization with IBM Composer

Objective: Set up a quantum circuit to simulate the QKD process.

Setup:

- Quantum Registers: Create a quantum register (qreg q[5]) to represent the 5 qubits used for communication between Alice and Bob.
- Classical Registers: Add two classical registers:
    - creg alice_bits[5] for Alice's measurement results.
    - creg bob_bits[5] for Bob's measurement results.

This forms the foundation for encoding, transmitting, and measuring quantum information.

Step 2: Alice's Encoding

Objective: Encode bits in qubits using random bases and values.

Hard-coded Encoding:

- **Qubit 0:**
    - Apply an X gate to flip the qubit and encode **bit 1** (remove the X gate for **bit 0**).
    - Apply an H gate to switch to the **diagonal basis** (remove the H gate to stay in the **computational basis**).
- **Qubit 1:**
    - Apply an H gate for encoding **bit 0** in the diagonal basis.
- **Qubit 2:**
    - Apply an X gate to encode **bit 1** in the computational basis (no H gate applied).
- **Qubit 3:**
    - Apply an X gate to encode **bit 1**.
    - Apply an H gate to switch to the diagonal basis.
- **Qubit 4:**
    - Apply an H gate to encode **bit 0** in the diagonal basis.

By the end of this step, Alice has encoded her qubits with specific values and bases.

Step 3: Bob's Measurement

Objective: Measure the incoming qubits using random bases.

Hard-coded Measurement Choices:

- Qubit 0:
    - No H gate applied, so Bob measures in the computational basis.
    - Result is stored in bob_bits[0].
- Qubit 1–4:
    - Similarly, no H gates are applied, meaning all measurements are in the computational basis.
    - Results are stored in respective bob_bits registers.

Bob now has his measurement results based on his chosen (and hard-coded) measurement bases.

Step 4: Optional Verification (Alice's Measurements)

Objective: Validate the protocol's behavior by comparing Alice's encoding directly.

- Alice measures her qubits in the basis she used for encoding:
    - Apply no additional gates before measuring.
    - Results are stored in alice_bits[0–4].

This step ensures that Alice's encoded values can be compared directly with Bob's measurements for debugging and validation purposes.

Step 5: Simulation and Output

Execution:

1. Run the Circuit:
    - Submit the circuit on a real quantum hardware.
2. Retrieve Results:
    - Extract measurement results from alice_bits and bob_bits classical registers.

Analysis:

- Compare Measurements:
    - Identify bits where Alice's encoding basis matches Bob's measurement basis.
    - Discard bits where the bases differ, as these results are unreliable.
- Post-Processing:
    - Use the matched bits to generate a shared secret key.
    - Optionally, check a subset of matched bits for consistency to detect potential eavesdropping.

7. RESULT AND DISCUSSION

Intermediatory stage wise debugging using inspect tool IBM Quantum Composer Circuit Simulation with Statevector and Q-sphere in a noise free environment.

First Step- Applying initial gates

The following first image shows the original state of this circuit. Alice prepares qubits 0, 2, and 3 in the computational basis by applying X where needed, and qubits 1 and 4 in the Hadamard basis by applying H gate. Then Bob will measure the qubits.

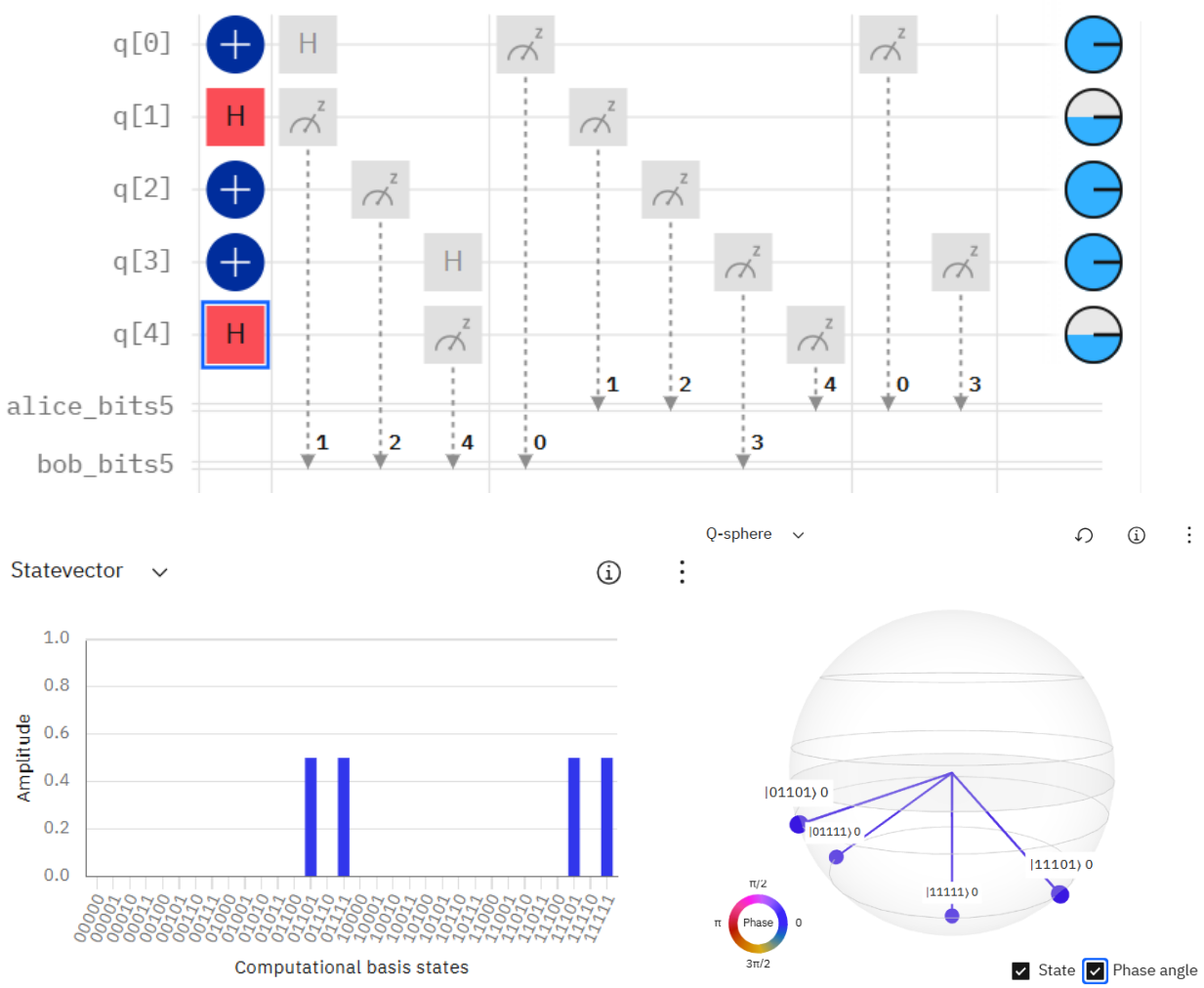

Fig 7.1 Applying initial gates

Next step- Superimposing and measuring Qbits

The second one, on qubit 4: Bob chose randomly the computational basis - the measurement in Z - thus different from the Hadamard basis in which Alice prepared it; this will likely result in a bit error or, equivalently, random measurement outcome for Bob.

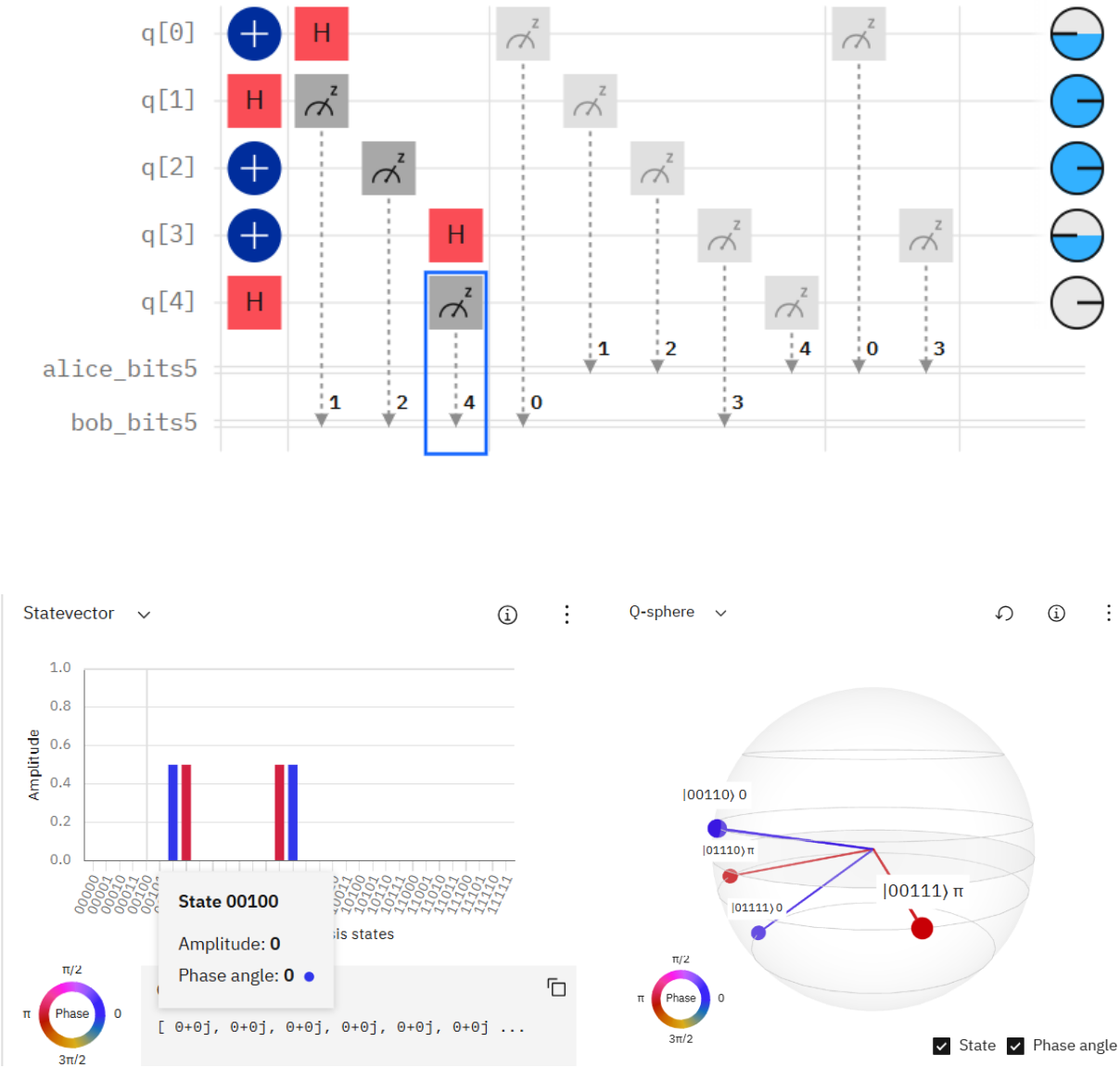

Fig 7.2 Superimposing and measuring Qbits

Next Step- Measuring the Qbits

The third image shows just another measurement, this time on qubit 3. Again, Bob has used a different basis than Alice, and so there is probably a bit error here, too. Fortunately, all of this is actually just part of the core steps in the BB84 protocol: Bob may measure in the wrong

basis any number of times, but subsequent error correction and privacy amplification will make up for what errors might have occurred.

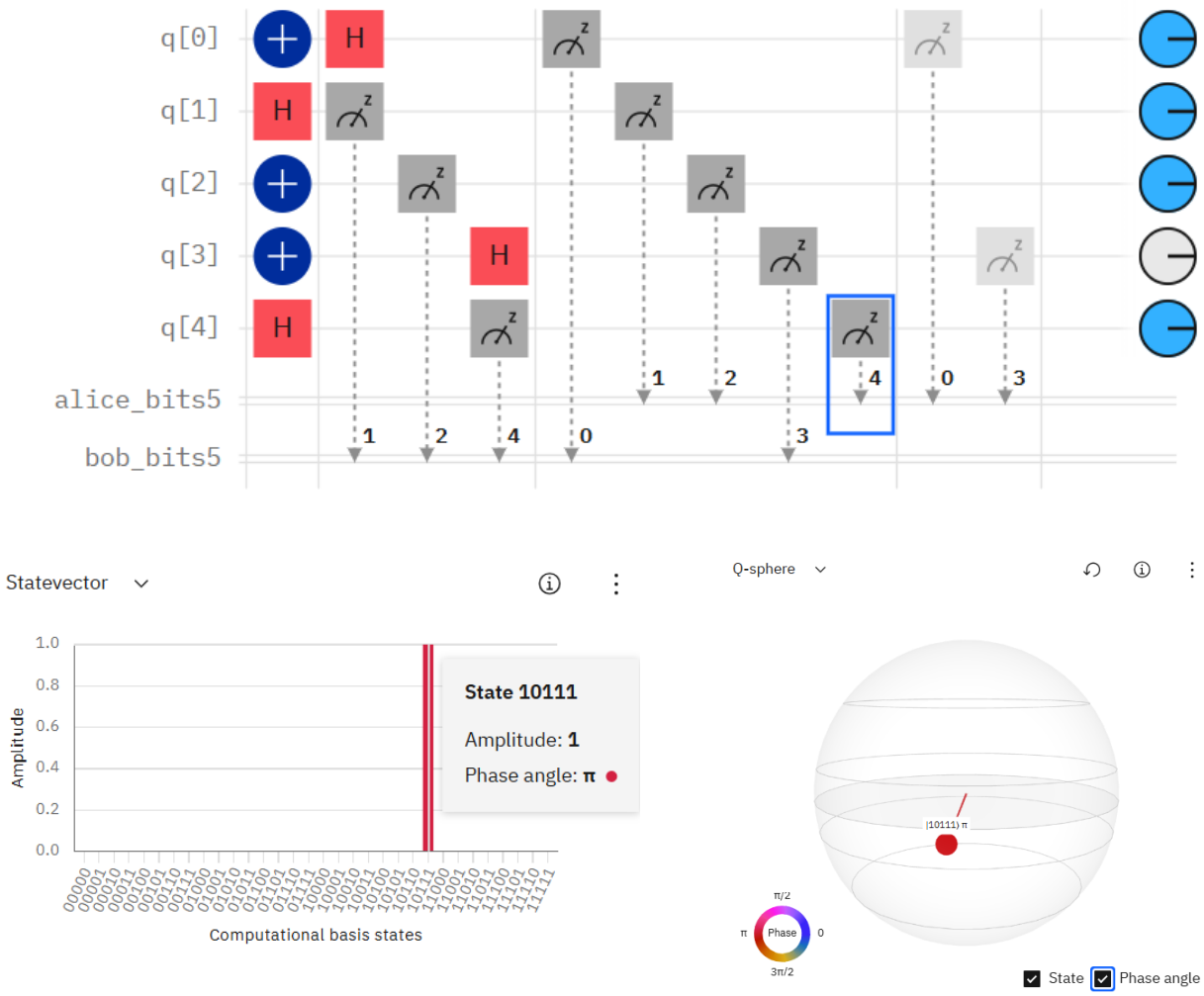

Fig 7.3 Measuring the Qubits

Final Step- Getting the final same keys for the users

The fourth image highlights the measurement of qubit 0. Similarly, if Bob chooses the wrong basis for his measurement (and the protocol doesn't have error correction), this could result in a mismatch between Alice and Bob's bits, highlighting the need for error correction in a real

BB84 implementation. The final measurements on each qubit are performed, and the results are saved in the classical registers.

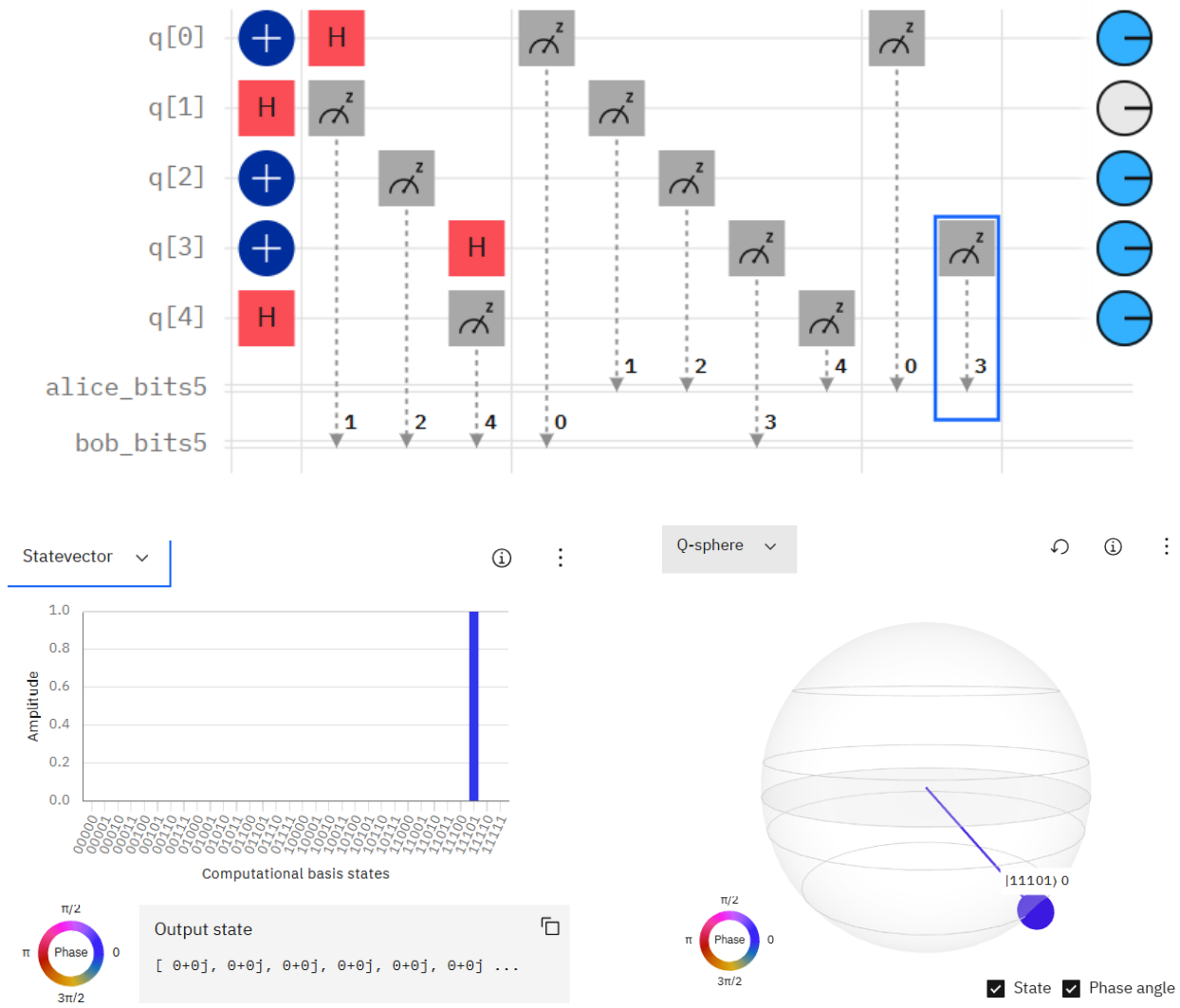

Fig 7.4 Getting the final same keys for the users

There is a statevector and a Q-sphere representation. This is the result of a single run of this 5-qubit BB84 simulation.

Statevector: A trivial bar chart of all 32 possible states of the five qubits (2^5=32). Virtually all the amplitude is concentrated at the last stage in one state (|11101>). This means the measured state will be this.

Q-Sphere: The Q-sphere is the final quantum state for the qubits. Most probably, one would find the qubits in state |11101> 0.

Randomness: The security of BB84 heavily depends on the randomness of Alice's bit choices and basis choices and also Bob's basis choices. It is an essential prerequisite to avoid the interception.

Basis Reconciliation: Alice and Bob publicly compare the bases they used for each qubit. They keep only those qubits where they used the same basis-that is called sifting.

Error correction and privacy amplification: These are not employed in the simplified QASM code of this answer. It can correct several of the errors that noise introduces into qubits, and it can ensure that an eavesdropper has learned nothing about the key.

8. CONCLUSION

8.1 Using IBM Composer

This job was executed on the IBM Brisbane quantum computer, **a 65-qubit device**. The submission from `*ibm-q/open/main*` indicates it was run on the publicly accessible IBM Quantum Experience. The results were generated using a sampler program, suggesting the system sampled the circuit's outcomes.

- **alice_bits Histogram:** This displays the distribution of Alice's measurements across her 5 qubits. The lack of uniformity in the distribution is expected, given the initial state and the Hadamard gates applied. (00101> with Highest Freq: 708/10000)

- **bob_bits Histogram:** This shows Bob's measurement distribution. The non-uniformity suggests one of two possibilities: (00111> Highest Freq: 695/10000)
  - The protocol may not employ a strictly random basis selection, or more likely,
  - Noise in the simulator or the execution is influencing the measurement results. Real quantum systems are subject to noise, leading to inconsistent outcomes.

Fig 8.1 Histogram for register "alice_bits" and "bob_bits"

While the histograms and circuit diagram provide a snapshot of the quantum computation, they don't represent a fully functional BB84 key distribution. The non-uniform distributions are likely due to noise, though they might also indicate a randomness issue in the protocol's implementation. In this case we prefer to use the keys which have attained the highest frequency from all the other keys, and treat it as the secret key for both the parties. Achieving

a correct and secure BB84 protocol would require addressing simultaneous measurement issues and incorporating basis reconciliation, error correction, and privacy amplification.

A comprehensive and meticulous diagrams that illustrates the various encoding and measurement processes concerning each qubit in the QKD Protocol and the details of the matrices in Alice's and Bob's coordinate systems in terms of those bases, bits values and whether they are corresponding or not.

Result Table for Matching bits:

Qubit	Alice's Basis	Alice's Bit	Bob's Basis (Assumed)	Bob's Bit	Result (Matched or Not)
0	Computational	0	Diagonal	1	**Mismatch**
1	Diagonal	0	Computational	1	**Mismatch**
2	Computational	1	Diagonal	1	**Match**
3	Diagonal	0	Computational	1	**Mismatch**
4	Computational	1	Diagonal	1	**Match**

Table 8.1 Results Table for Matching Bits

Key Generation Process:

When comparing the key generation process between Alice and Bob it indicates that Qubits 2 and 4 are matched (i.e. they show the same bit value when measured in the same basis). When considering the generation of a key however Qubits 0, 1, and 3 cannot be used as they show different bits or bases.

Final key Generation Process:

Alice's key consists of 1 which is contributed by Qubit 2 and also 1 contributed by Qubit 4.

Bob's key contains 1 from Qubit 2 and 1 from Qubit 4 as well.

In short : the shared final key = 11.

Alice and Bob check for bases after they communicate the message. The only pieces of information that are useful are the bits where their bases were equal. Therefore, in this case, bits at positions 2 and 4 of the final secret key are formed as both Alice and Bob applied the same basis when measuring those bits.

8.2 Using Qiskit Aer-Simulator on Python –

This QKD simulation performs an idealized version of the BB84 protocol. It aims at visualizing the secure key exchange between two parties, identified as Alice and Bob, using quantum mechanics. The bits are encoded in computational or diagonal bases, chosen at random. Alice encodes her bits into qubits and applies a Hadamard gate on the ones in the diagonal basis. Bob then prepares the measurement bases randomly without knowing which bases Alice actually used, and records his measurement results.

```
--- QKD Simulation Details ---
Alice's Bits:        [1, 1, 1, 1, 0]
Alice's Bases:       ['diagonal', 'computational', 'diagonal', 'diagonal', 'diagonal']
Bob's Bases:         ['diagonal', 'computational', 'diagonal', 'computational', 'diagonal']
Bob's Bits:          [0, 1, 0, 1, 0]

Matched Indices:     [1, 4]
Mismatched Indices:  [0, 2, 3]

Final Key:           10
```

Fig 8.2 QKD Simulation Details

Qubit	Alice's Basis	Alice's Bit	Bob's Basis (Assumed)	Bob's Bit	Result (Matched or Not)
0	Diagonal	1	Diagonal	0	**Mismatch**
1	Computational	1	Computational	1	**Match**
2	Diagonal	1	Diagonal	0	**Mismatch**
3	Diagonal	1	Computational	1	**Match**
4	Diagonal	0	Diagonal	0	**Mismatch**

Table 8.2 Results using Qiskit Aer-Simulator on Python

Results include matched and mismatched bases. The indices at which Alice's and Bob's bases align would indicate positions where their bits should, in theory, match and thus form the shared secret key. Mismatched indices show the randomness and privacy guaranteed by the protocol, as these are discarded. This empirical key is based purely on the matched bases, proof that, even without any prior information, a secure key can be distilled resulting from the principles of quantum mechanics. Had an eavesdropper tried to intercept the qubits, their presence would have disturbed the system by introducing errors and hence revealing the intrusion. The simulation gives good insight into the very basic principle of QKD: the enabling of secure communication by quantum properties.