

Name: Gudur Krishna Chaitanya Roll no: 102117049 email used for
gchaitanya_be21@thapar.edu

BB84 algorithm:

Problem: The classical problem of securely distributing cryptographic keys over an insecure channel is susceptible to eavesdropping, interception, and manipulation. The BB84 protocol aims to overcome these issues using the principles of quantum mechanics.

Quantum Properties: The security of the BB84 protocol relies on two fundamental properties of quantum mechanics:

Superposition: Quantum bits (qubits) can exist in multiple states simultaneously, enabling the transmission of information in a superposition of states.

Uncertainty Principle: Measurement of a quantum state disturbs the state, making it impossible to measure the complete information of a qubit without disturbing it.

Key Steps:
Step 1: Key Generation: Alice prepares a random sequence of qubits, encoding each qubit in one of two bases (typically the computational basis $|0\rangle$ and $|1\rangle$, or the Hadamard basis $|+\rangle$ and $|-\rangle$).

Step 2: Transmission: Alice sends the encoded qubits to Bob over the quantum channel. Due to the uncertainty principle, an eavesdropper (traditionally named Eve) cannot measure the complete information of a qubit without disturbing it.

Step 3: Measurement: Upon receiving the qubits, Bob randomly selects a basis for each qubit and measures it accordingly.

Step 4: Error Correction: Alice and Bob publicly announce the bases used for each qubit. They discard qubits measured in different bases and use the remaining qubits to generate a preliminary key.

Step 5: Privacy Amplification: To ensure security against eavesdropping, Alice and Bob perform privacy amplification techniques (such as hash functions) to distill a final secret key from the preliminary key.

Security: The security of the BB84 protocol relies on the principles of quantum mechanics. Any attempt by Eve to intercept or measure the qubits will inevitably introduce errors, which can be detected by Alice and Bob during the key reconciliation phase.

```
1 from qiskit_ibm_provider import IBMProvider
2
3 provider = IBMProvider(token='f55702335547d565b44eb80fd6708f3b82b4d0147236062')
4
5 active_account = provider.active_account()
6
7 print("Active Account Details:")
8
9 print(active_account)
```

```
Active Account Details:
{'channel': 'ibm_quantum', 'token': 'f55702335547d565b44eb80fd6708f3b82b4d0147236062'}
```

```
1 from qiskit import QuantumCircuit, transpile
```

```

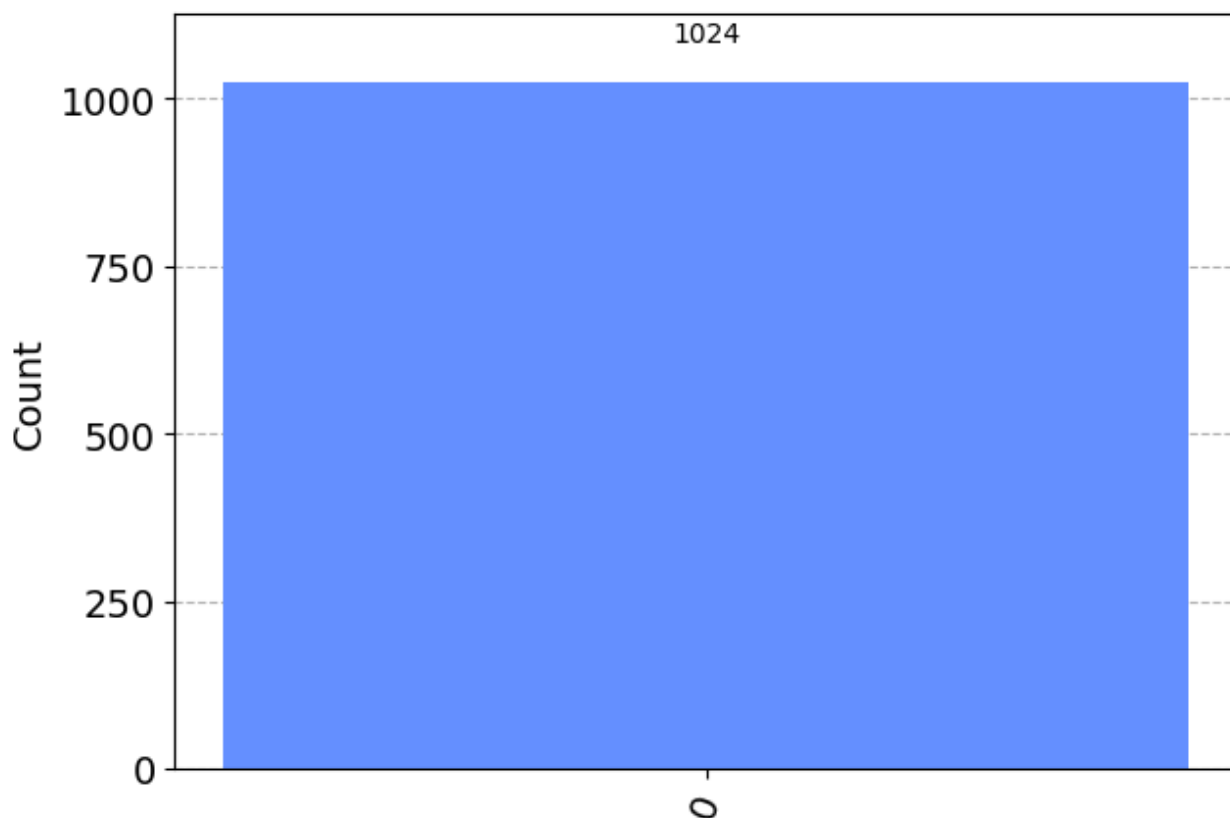
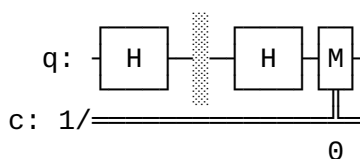
1 from qiskit import QuantumCircuit, transpile
2 import qiskit_aer
3 from qiskit.visualization import plot_histogram, plot_bloch_multivector
4 from numpy.random import randint
5 import numpy as np

```

```

1 qc = QuantumCircuit(1,1)
2 qc.h(0)
3 qc.barrier()
4 qc.h(0)
5 qc.measure(0,0)
6 display(qc.draw(output='text',style='bw'))
7 aer_sim = qiskit_aer.Aer.get_backend('aer_simulator')
8 job = aer_sim.run(qc)
9 plot_histogram(job.result().get_counts())

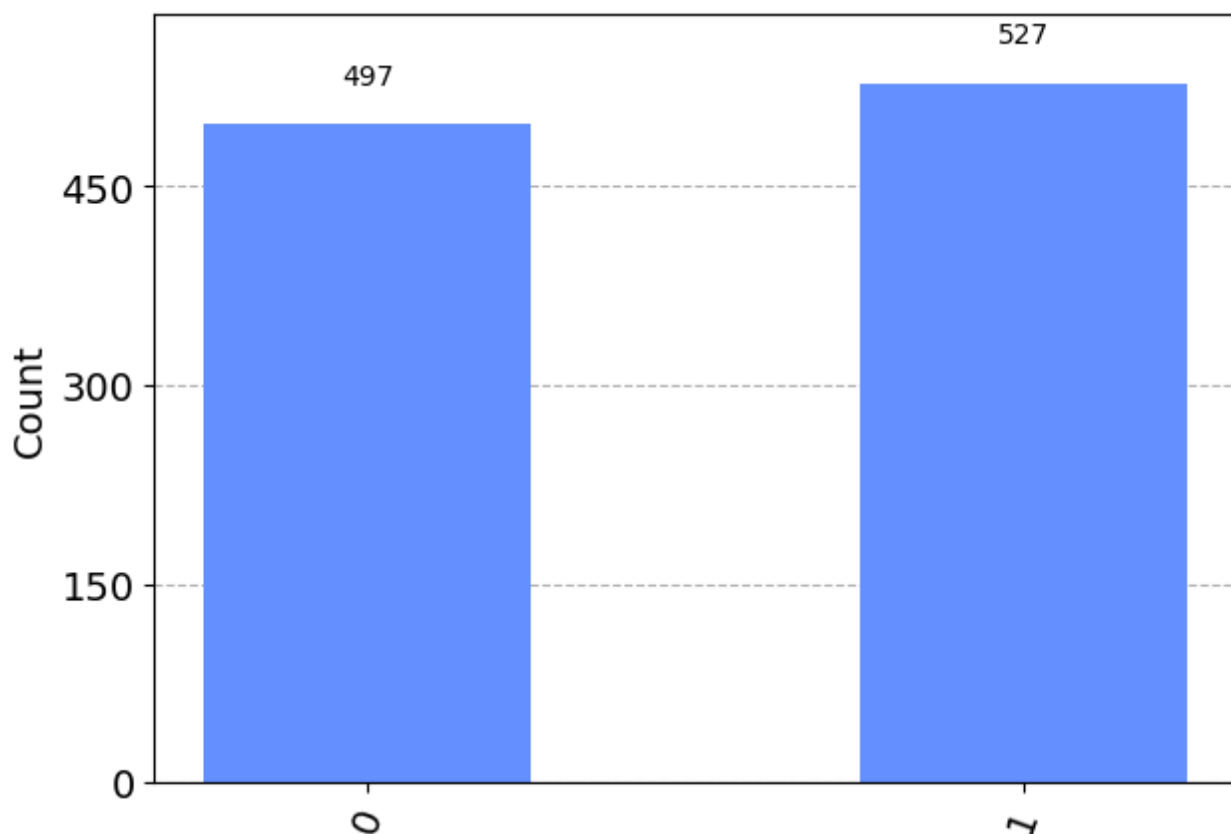
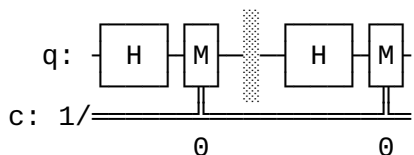
```



```

1 qc = QuantumCircuit(1,1)
2 qc.h(0)
3 qc.measure(0, 0)
4 qc.barrier()
5 qc.h(0)
6 qc.measure(0,0)
7 display(qc.draw(output='text',style='bw'))
8 aer_sim = qiskit_aer.Aer.get_backend('aer_simulator')
9 job = aer_sim.run(qc)
10 plot_histogram(job.result().get_counts())

```



```
1 np.random.seed(seed=0)
```

```
1 n = 100
```

```
1 np.random.seed(seed=0)
```

```
2 n = 100
```

```
3 alice_bits = randint(2, size=n)
```

```
4 print(alice_bits)
```

```
[0 1 1 0 1 1 1 1 1 1 1 0 0 1 0 0 0 0 0 1 0 1 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0
 1 1 0 0 1 0 1 1 1 1 1 0 1 0 1 1 1 1 0 1 0 0 1 1 0 1 0 1 0 0 0 0 1 1 0 0
 0 1 1 0 1 0 0 1 0 1 1 1 1 1 1 0 1 1 0 0 1 0 0 1 1 0]
```

```
1 np.random.seed(seed=0)
```

```
2 n = 100
```

```
3 alice_bits = randint(2, size=n)
```

```
4 alice_bases = randint(2, size=n)
```

```
5 print(alice_bases)
```

```
[1 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1 1 0 1 1 0 0 1 0
 0 0 0 1 1 0 0 1 0 1 1 1 1 0 0 0 1 0 1 1 1 0 1 0 0 1 0 1 1 0 0 1 0 1 0 1 0
 1 0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0]
```

```
1 def encode_message(bits, bases):
```

```

2     message = []
3     for i in range(n):
4         qc = QuantumCircuit(1,1)
5         if bases[i] == 0:
6             if bits[i] == 0:
7                 pass
8             else:
9                 qc.x(0)
10        else:
11            if bits[i] == 0:
12                qc.h(0)
13            else:
14                qc.x(0)
15                qc.h(0)
16        qc.barrier()
17        message.append(qc)
18    return message

1 np.random.seed(seed=0)
2 n = 100
3 alice_bits = randint(2, size=n)
4 alice_bases = randint(2, size=n)
5 message = encode_message(alice_bits, alice_bases)

```

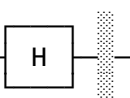
```

1 print('bit = %i' % alice_bits[0])
2 print('basis = %i' % alice_bases[0])

    bit = 0
    basis = 1

```

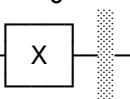
```
1 message[0].draw(output='text', style='bw')
```

q: 
c: 1/

```

1 print('bit = %i' % alice_bits[4])
2 print('basis = %i' % alice_bases[4])
3 message[4].draw(output='text', style='bw')

```

bit = 1
basis = 0
q: 
c: 1/

```

1 np.random.seed(seed=0)
2 n = 100
3 alice_bits = randint(2, size=n)
4 alice_bases = randint(2, size=n)
5 message = encode_message(alice_bits, alice_bases)
6 bob_bases = randint(2, size=n)

```

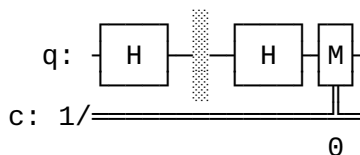
```
7 print(bob_bases)
```

```
[1 0 1 0 0 1 1 0 0 0 1 1 0 0 0 0 0 1 0 1 0 0 0 1 1 1 0 0 1 1 1 1 0 0 0 1 1
 0 1 0 0 1 0 1 1 1 1 0 0 0 1 1 1 0 1 1 1 1 0 0 1 1 0 0 0 1 1 0 1 1 1 1 1 0
 0 0 1 0 1 0 1 1 0 0 0 1 0 0 1 1 1 1 0 1 0 0 0 0 1 1]
```

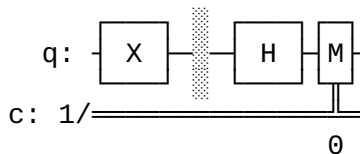
```
1 def measure_message(message, bases):
2     backend = qiskit_aer.Aer.get_backend('aer_simulator')
3     measurements = []
4     for q in range(n):
5         if bases[q] == 0:
6             message[q].measure(0,0)
7         if bases[q] == 1:
8             message[q].h(0)
9             message[q].measure(0,0)
10    aer_sim = qiskit_aer.Aer.get_backend('aer_simulator')
11    result = aer_sim.run(message[q], shots=1, memory=True).result()
12    measured_bit = int(result.get_memory()[0])
13    measurements.append(measured_bit)
14    return measurements
```

```
1 np.random.seed(seed=0)
2 n = 100
3 alice_bits = randint(2, size=n)
4 alice_bases = randint(2, size=n)
5 message = encode_message(alice_bits, alice_bases)
6 bob_bases = randint(2, size=n)
7 bob_results = measure_message(message, bob_bases)
```

```
1 message[0].draw(output='text', style='bw')
```



```
1 message[6].draw(output='text', style='bw')
```



```
1 print(bob_results)
```

```
[0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0]
```

```
1 def remove_garbage(a_bases, b_bases, bits):
2     good_bits = []
3     for q in range(n):
4         if a_bases[q] == b_bases[q]:
5             good_bits.append(bits[q])
6     return good_bits
```

```

1 np.random.seed(seed=0)
2 n = 100
3 alice_bits = randint(2, size=n)
4 alice_bases = randint(2, size=n)
5 message = encode_message(alice_bits, alice_bases)
6 bob_bases = randint(2, size=n)
7 bob_results = measure_message(message, bob_bases)
8 alice_key = remove_garbage(alice_bases, bob_bases, alice_bits)
9 print(alice_key)

[0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1,

```

```

1 np.random.seed(seed=0)
2 n = 100
3 alice_bits = randint(2, size=n)
4 alice_bases = randint(2, size=n)
5 message = encode_message(alice_bits, alice_bases)
6 bob_bases = randint(2, size=n)
7 bob_results = measure_message(message, bob_bases)
8 alice_key = remove_garbage(alice_bases, bob_bases, alice_bits)
9 bob_key = remove_garbage(alice_bases, bob_bases, bob_results)
10 print(bob_key)

[0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1,

```

```

1 def sample_bits(bits, selection):
2     sample = []
3     for i in selection:
4         i = np.mod(i, len(bits))
5         sample.append(bits.pop(i))
6     return sample

```

```

1 np.random.seed(seed=0)
2 n = 100
3 alice_bits = randint(2, size=n)
4 alice_bases = randint(2, size=n)
5 message = encode_message(alice_bits, alice_bases)
6 bob_bases = randint(2, size=n)
7 bob_results = measure_message(message, bob_bases)
8 alice_key = remove_garbage(alice_bases, bob_bases, alice_bits)
9 bob_key = remove_garbage(alice_bases, bob_bases, bob_results)
10 sample_size = 15
11 bit_selection = randint(n, size=sample_size)
12 bob_sample = sample_bits(bob_key, bit_selection)
13 print("  bob_sample = " + str(bob_sample))
14 alice_sample = sample_bits(alice_key, bit_selection)
15 print("alice_sample = "+ str(alice_sample))

    bob_sample = [0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
    alice_sample = [0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]

```

```

1 bob_sample == alice_sample

True

```

```

1 print(bob_key)
2 print(alice_key)
3 print("key length = %i" % len(alice_key))
    [1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0,
    [1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0,
    key length = 33

```

```
1 np.random.seed(seed=3)
```

```

1 np.random.seed(seed=3)
2 alice_bits = randint(2, size=n)
3 print(alice_bits)
    [0 0 1 1 0 0 0 1 1 1 0 1 1 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0 1 1 0 1
    0 0 1 1 0 0 1 0 1 0 1 1 1 1 0 1 0 0 1 1 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 1 1
    1 0 1 1 1 1 1 1 0 1 0 0 1 1 0 1 0 0 0 0 0 1 1 0 1 1]

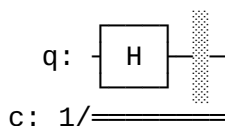
```

```

1 np.random.seed(seed=3)
2 alice_bits = randint(2, size=n)
3 alice_bases = randint(2, size=n)
4 message = encode_message(alice_bits, alice_bases)
5 print(alice_bases)
    [1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 0 0 1 1 0 0 1 1 1 1 0 0
    1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 1 1 0 0 1 0 0 1 1 1 1
    1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 0 0 0 1 0 1 1 1]

```

```
1 message[0].draw(output='text',style='bw')
```

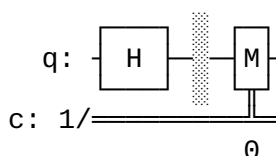


```

1 np.random.seed(seed=3)
2 alice_bits = randint(2, size=n)
3 alice_bases = randint(2, size=n)
4 message = encode_message(alice_bits, alice_bases)
5 eve_bases = randint(2, size=n)
6 intercepted_message = measure_message(message, eve_bases)
7 print(intercepted_message)
    [1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,

```

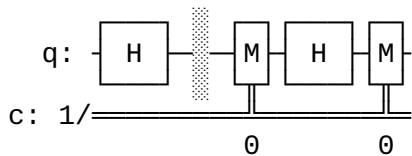
```
1 message[0].draw(output='text',style='bw')
```



```

1 np.random.seed(seed=3)
2 alice_bits = randint(2, size=n)
3 alice_bases = randint(2, size=n)
4 message = encode_message(alice_bits, alice_bases)
5 eve_bases = randint(2, size=n)
6 intercepted_message = measure_message(message, eve_bases)
7 bob_bases = randint(2, size=n)
8 bob_results = measure_message(message, bob_bases)
9 message[0].draw(output='text', style='bw')

```



```

1 np.random.seed(seed=3)
2 alice_bits = randint(2, size=n)
3 alice_bases = randint(2, size=n)
4 message = encode_message(alice_bits, alice_bases)
5 eve_bases = randint(2, size=n)
6 intercepted_message = measure_message(message, eve_bases)
7 bob_bases = randint(2, size=n)
8 bob_results = measure_message(message, bob_bases)
9 bob_key = remove_garbage(alice_bases, bob_bases, bob_results)
10 alice_key = remove_garbage(alice_bases, bob_bases, alice_bits)

```

```

1 np.random.seed(seed=3)
2 alice_bits = randint(2, size=n)
3 alice_bases = randint(2, size=n)
4 message = encode_message(alice_bits, alice_bases)
5 eve_bases = randint(2, size=n)
6 intercepted_message = measure_message(message, eve_bases)
7 bob_bases = randint(2, size=n)
8 bob_results = measure_message(message, bob_bases)
9 bob_key = remove_garbage(alice_bases, bob_bases, bob_results)
10 alice_key = remove_garbage(alice_bases, bob_bases, alice_bits)
11 sample_size = 15
12 bit_selection = randint(n, size=sample_size)
13 bob_sample = sample_bits(bob_key, bit_selection)
14 print("  bob_sample = " + str(bob_sample))
15 alice_sample = sample_bits(alice_key, bit_selection)
16 print("alice_sample = " + str(alice_sample))

```

```

    bob_sample = [1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0]
    alice_sample = [1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]

```

```

1 bob_sample == alice_sample
False

```

```

1 n = 100
2 alice_bits = randint(2, size=n)
3 alice_bases = randint(2, size=n)

```



```
4 message = encode_message(alice_bits, alice_bases)
5 eve_bases = randint(2, size=n)
6 intercepted_message = measure_message(message, eve_bases)
7 bob_bases = randint(2, size=n)
8 bob_results = measure_message(message, bob_bases)
9 bob_key = remove_garbage(alice_bases, bob_bases, bob_results)
10 alice_key = remove_garbage(alice_bases, bob_bases, alice_bits)
11 sample_size = 15
12 bit_selection = randint(n, size=sample_size)
13 bob_sample = sample_bits(bob_key, bit_selection)
14 alice_sample = sample_bits(alice_key, bit_selection)
15 if bob_sample != alice_sample:
16     print("Eve's interference was detected.")
17 else:
18     print("Eve went undetected!")
    Eve's interference was detected.
```