

Report

on

Under the supervision
of
Asst. Prof. Dr. Aswath Babu

Course :- EC357
Semester :- 5th
Course Instructor :- Asst. Prof. Dr. Mukesh Mishra

Team-3

Aalekh Prasad	-	20BEC001
Gaurang Belekar	-	20BEC015
Neel Shahakar	-	20BEC026
Rujula Karanjkar	-	20BEC041
Virat Rajat Nayak	-	20BEC058

INTRODUCTION :

- **Quantum communication**

Usually sensitive data/information is encrypted and sent across fiber-optic cables and other alternative channels along with the digital keys required to decipher the data. The information and the keys are sent as classical bits—a stream of electrical or optical pulses representing 1s and 0s. And that makes the data vulnerable while communicating. Security threats can scan and duplicate bits in transit while not featuring a trace. This makes intricate sectors like banking and blockchains unsafe.

Quantum communication uses the laws of quantum physics to secure data. These laws permit particles, typically bosons transmitting data along optical cables to take on a state of superposition, which means they can be represented by multiple combinations of 1 and 0 simultaneously. These particles are known as quantum bits, or qubits.

These qubits collapse from their fragile quantum superposition states to classical states 1 or 0 on classical measurement. Thus an eavesdropper cannot extract and duplicate bits in transit while not featuring a trace.

Quantum communication paves way for cryptographic security of information using Quantum key distribution protocols.

- **Quantum Key Distribution**

Quantum Key Distribution (QKD) is a secure communication methodology for exchanging encrypted keys only known between shared parties. The communication method uses properties found in quantum physics to exchange cryptographic keys in a way that can be proved to guarantee security.

QKD enables two parties to produce and share a key that is then used to encrypt and decrypt messages, it involves sending encrypted data as classical bits over networks, while the keys to decrypt the information are encoded and transmitted in a quantum state using qubits. Various approaches, or protocols, have been developed for implementing this.

The two main categories are prepare-and-measure protocols and Entanglement-based protocols. Prepare-and-measure protocols focus on measuring unknown quantum states. This type of protocol can be used to detect eavesdropping, as well as how much data was potentially intercepted. Entanglement-based protocols focus around quantum states in which two nodes are linked together to form a quantum network using a combined quantum state. The concept of entanglement means that measurement of one object thereby affects the other. In this method, if an eavesdropper accesses a previously trusted node and changes something, the other involved parties will be affected.

A widely used QKD technique is the BB84 Protocol which will be used in the project.

- **BB84 protocol**

A user A (Alice) wants to send data securely to user B (Bob). To do so an encryption key is created in the form of qubits whose polarization states represent the individual bit values of the key. The qubits can be sent to Bob through a fiber-optic cable. By comparing measurements of the state of a fraction of these qubits (key sifting) Alice and Bob can establish that they hold the same key.

As the qubits travel to their destination, the fragile quantum state of some of the qubits will collapse because of decoherence. To account for this, Alice and Bob run through the KEY DISTILLATION PROCESS, which involves calculating whether the error rate is high enough to

suggest that an eavesdropper has tried to intercept the key. If it is, they eliminate the suspect key and keep generating new ones until they are confident that they share a secure key. Both parties can then use this key for bidirectional encrypted classical communication.

- **Qiskit:**

Qiskit is an open-source software development kit (SDK) for working with quantum computers at the level of circuits, pulses, and algorithms. Qiskit consists of elements that work together to perform quantum computation.

Qiskit was founded by IBM Research to allow software development for their cloud quantum computing service, IBM Quantum Experience. Contributions are also made by external supporters, typically from academic institutions.

- **Netsquid:**

The Network Simulator for Quantum Information using Discrete Events (NetSquid) is a software tool for the modeling and simulation of scalable quantum networks developed at QuTech laboratories. The goal of NetSquid is to enable scientists and engineers to design the future quantum internet as well as modular quantum computing architectures.

One of NetSquid's key features is its ability to easily and accurately model the effects of time on the performance of quantum network and quantum computing systems. NetSquid's modular approach allows a detailed physical modeling of individual components

Objectives:

Comparing BB84 protocol on Qiskit Simulator vs Parameterised Noisy Network Simulator, NetSquid which simulates real world quantum communication networks.

Quantum Noisy Simulation:

Working:

The first step is for Alice to choose a string of bits and bases to encode them in. It is done by choosing any length of binary string of any length. In our code we have generated a random key of length of 500 at the Alice side.

Based on the random key generation algorithm, random measurement bases are generated. This generated random bases is used to encode the generated key as follows:

Bits	Qubit	Basis	Gate Required
0	$ 0\rangle$	Z	None
1	$ 1\rangle$	Z	X
0	$ +\rangle$	X	H
1	$ -\rangle$	X	XH

In this BB84 simulation using qiskit, the quantum channel will just be an array which will store the encoded qubits which will be sent in simulation to Bob in order to be measured by him.

Here the noisy Quantum Computer is being simulated with measurement error on the gate implementation.

Alice needs to send Bob bases which she chose to encode her qubits in. She can tell him over *any* classical channel. In this protocol it doesn't matter if Eve (any external mediator) finds out which bases Alice used.

For each of the qubits where Alice and Bob chose different bases, there is a 50% chance Bob's measurement returned the wrong qubit. For example, If Alice sent Bob a qubit in the $|+\rangle$ state (i.e., a bit value of 0 encoded in the X-basis), and Bob measures in the Z-basis, there is a 50% chance he will get a $|0\rangle$ and a 50% chance he will get a $|1\rangle$.

Thus, every instance where their bases don't match is useless to them, so Bob needs to find the bases they share in common. As Bob knows the bases they share in common, he can discard the rest bits and keep the bits with the same bases.

The matched bits with the same bases will be the key on encryption between the two nodes of the channel (Alice and Bob).

Result:

Time taken for generation of symmetric key = 19.221s

```
<frozen importlib_bootstrap>:219: RuntimeWarning: scipy.lib.messagestream.MessageStream size changed, may indicate binary incompatibility. Expected 56 from C header, got 64 from PyObject

The bits Alice is going to send are: 1101111110...
The bases Alice is going to encode them in are: ZXZZZZZX...

q: [X]
c: _____

q: [X] [H]
c: _____

q: [H]
c: _____

q: [X]
c: _____

q: [X]
c: _____

etc.
The bases Bob is going to decode them in are: XZXZZZX...
The first few bits Bob received are: 0110111100...
The indices of the first 10 bases they share in common are: [4, 5, 6, 7, 9, 10, 11, 19, 22, 24]
Yep, Alice and Bob seem to have the same bits!
The Private shared key is
010101101001011101100110000011010100010101110100100011010100111000110000110011010010000
The key is 123 bits long.
Time taken to generate the key and transfer: 19.221593618392944
```

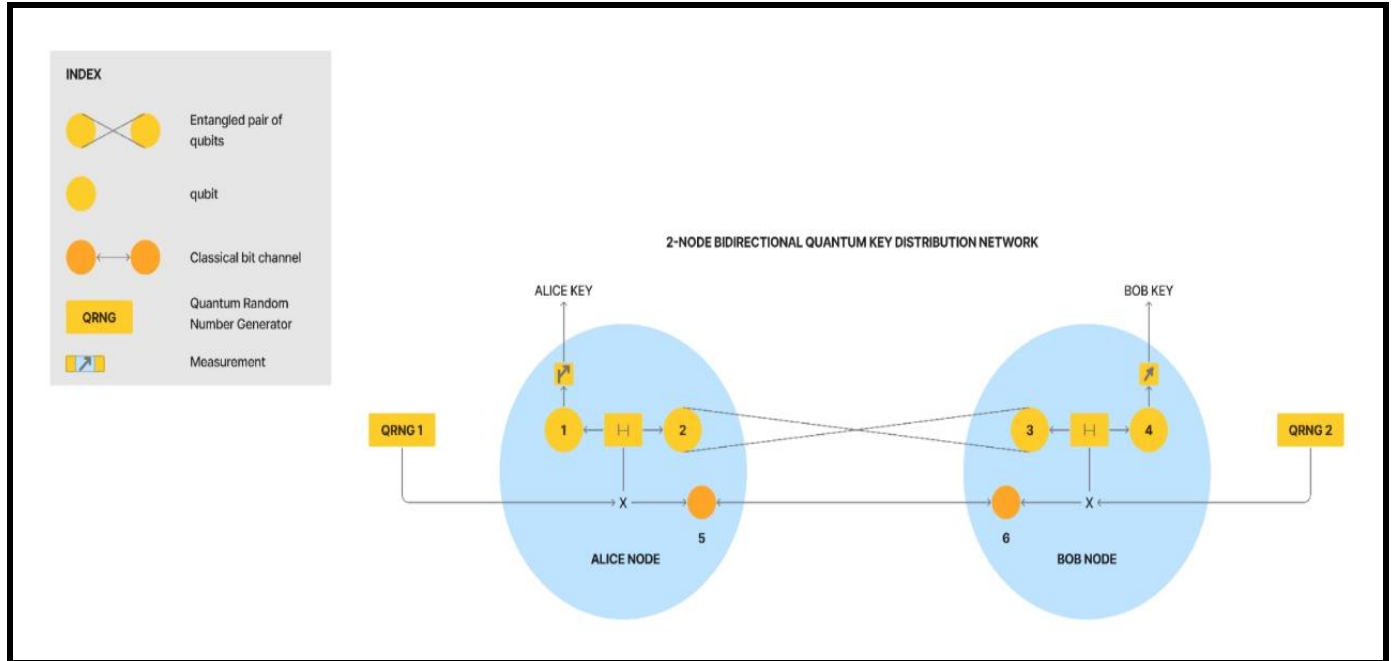
Inference:

The key generated between Alice and Bob is through a quantum communication channel in the most ideal state where there is no loss due to error in propagation but considering error in Quantum Computer due to measurement.

NOISY NETWORK SIMULATION:

In addition to simulation of the architecture of the BB84 protocol on Qiskit, the Quantum network simulator Netsquid will recreate the working of the complete communication network and the quantum and classical systems required for carrying out cryptographic implementation.

For this a simple two node hybrid Network (quantum + classical) was created. A single quantum channel consisting of bell-state entangled qubits and a bidirectional classical information channel was connected between these two nodes A and B.

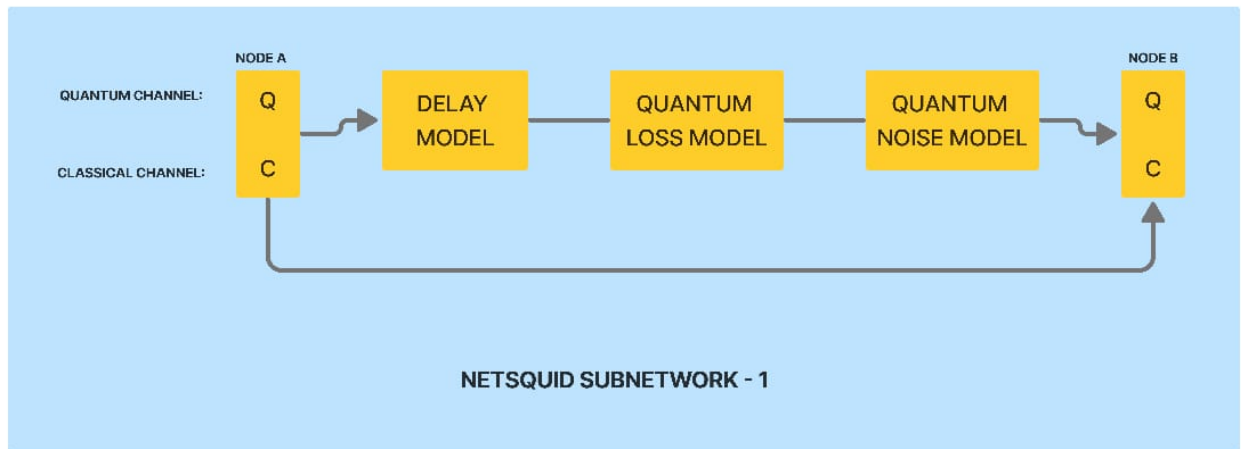


Each Node was connected to a simulation variant of QRNG (Quantum Random Number Generator) simulated in netsquid by a seed based Pseudo Random number generator interface with both the quantum and classical subsystems. The classical channel is connected to an n-qubit cryptographic quantum circuit which is further interfaced with the classical channel. A measurement channel estimates the quantum output into classical keys.

The classical channel bit width is set equal to the required key length with linear bit transfer delay.

The Quantum channel is passed through three real world attenuation models:-

- A. Delay Model - Delay caused by quantum physical processes in the quantum channel.
- B. Quantum Fiber Loss Model - loss of quantum information due to physical phenomena like bending and absorption of optical fiber.
- C. Quantum Noise Model - depolarisation caused by external environmental decoherence which couples the qubits to other degrees of freedoms.



Network code snippet :

```
n = 10000

def protocol(a):
    if __name__ == "__main__":
        alice_qmemory = QuantumMemory("Alice_Memory", num_positions=n, models={'delay_model': FixedDelayModel({)})
        bob_qmemory = QuantumMemory("Bob_Memory", num_positions=n, models={'delay_model': FixedDelayModel({)})

        alice = Node("Alice", qmemory=alice_qmemory, port_names=["qout_bob", "cin_bob"])
        bob = Node("Bob", qmemory=bob_qmemory, port_names=["qin_alice", "cout_alice"])

        channel_a2b = CombinedChannel("QC_Channel_a2b", length=130, models={"delay_model": FibreDelayModel(), "quantum_loss_model": FibreLossModel(p_loss_init=0, p_loss_length=0.2)}
        channel_b2a = CombinedChannel("QC_Channel_b2a", length=130, models={"delay_model": FibreDelayModel(), "quantum_loss_model": FibreLossModel(p_loss_init=0, p_loss_length=0.2)}
        connect = DirectConnection("Connection", channel_AtoB=channel_a2b, channel_BtoA=channel_b2a)

        network = Network(name="Network")
        network.add_nodes([alice, bob])
        network.add_connection(alice, bob, connection=connect, label="quantum", port_name_node1="qout_bob", port_name_node2="qin_alice")

        alice_protocol = AliceProtocol(alice, "qout_bob", channel_a2b)
        bob_protocol = BobProtocol(bob, "qin_alice")

        alice_protocol.receiver_protocol = bob_protocol
        bob_protocol.sender_protocol = alice_protocol

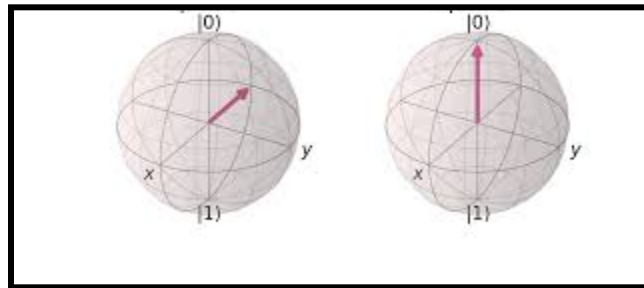
        key_bit_error=[]
        for j in range(1):
            ns.sim_reset()
            alice_protocol.start()
            bob_protocol.start()
            stats = ns.sim_run()
            list_length = getattr(bob_protocol, 'list_length')
            alice_matching_key = getattr(alice_protocol, 'matching_keybits')
            bob_matching_key = getattr(bob_protocol, 'matching_keybits')
            alice_key = [value for value in alice_matching_key.values()]
            bob_key = [value for value in bob_matching_key.values()]
            error_bits = list(map(lambda x, y: x ^ y, alice_key, bob_key))
            key_bit_error = (np.sum(error_bits) / list_length)

        print(f"The time required to establish the key is {ns.sim_time()}\n")
        print(f"The matched key according to Alice is:\n {alice_matching_key}\n")
        print(f"The matched key according to bob is:\n {bob_matching_key}\n")
        print(f"The matched key according to Alice is:\n {alice_key}\n")
        print(f"The matched key according to Alice is:\n {bob_key}\n")
        print(f"The key bit error for an iteration. is:\n {key_bit_error}\n")

#standard model: 100 0.2 1e3
```

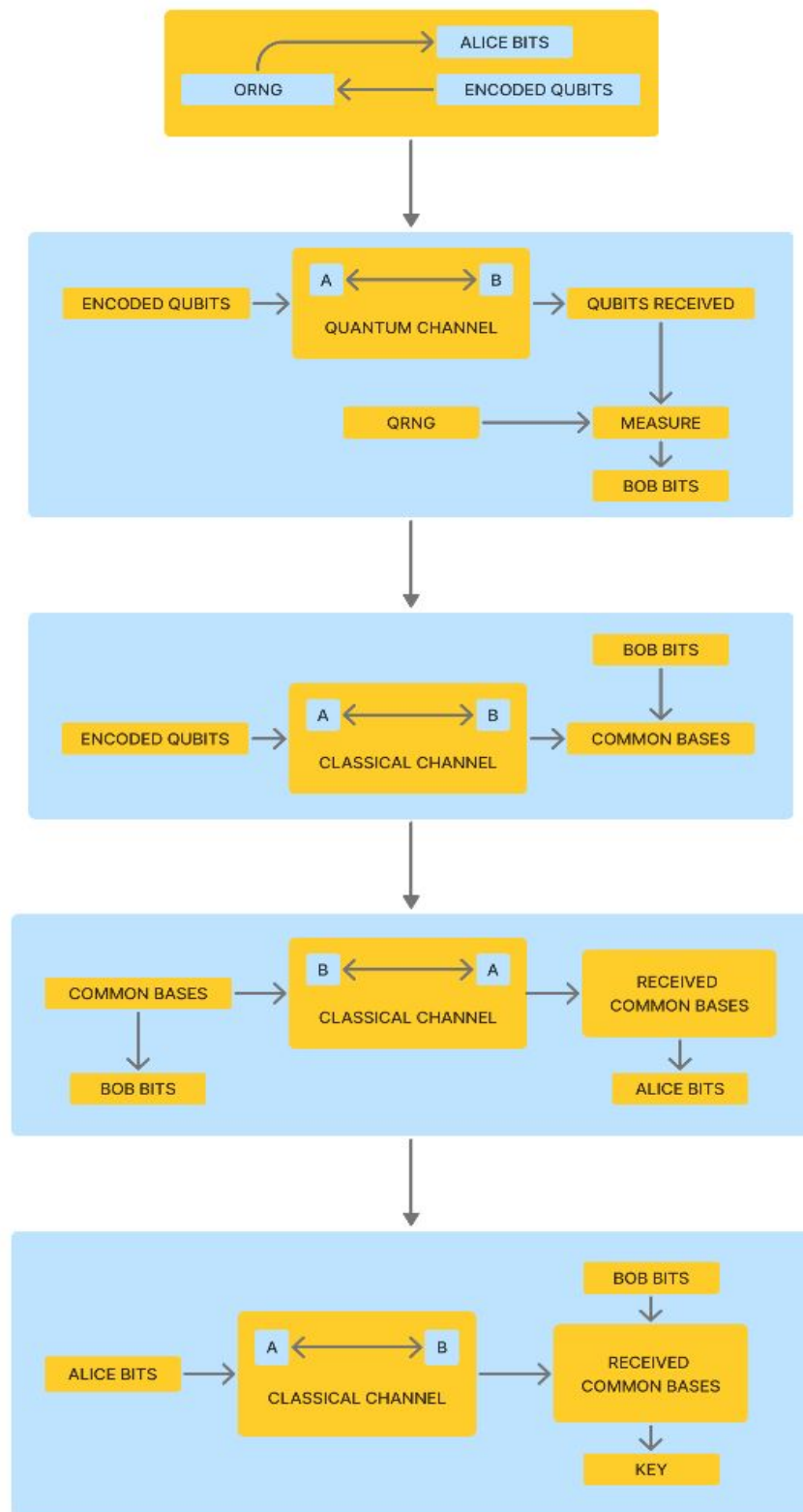
The protocol design workflow is as follows:-

- I. At node A, Alice uses a quantum random number generator to create a random bit array of given key length. This bit sequence is then encoded into orthogonal quantum states Plus-minus ket state and the zero - one ket state according to another random bit sequence generated by a QRNG.



- II. Then the encoded qubits are sent into the quantum channel according to a predefined time stamp. These qubits are received by bob at node B. They are then state decoded according to another random bit sequence generated by a QRNG and then Classically measured using the measurement channel to give bob's bits
- III. Alice's bases are sent through the classical channel to node B which are compared to bob's bases and the matching bases are recorded as COMMON BASES.
- IV. The Common bases are used by bob to rectify bob's bits. The Common bases are also sent to node A through the classical channel in order to rectify alice's bits by removing the non matching bases bits.
- V. The rectified Alice's bits are then sent through the quantum channel to node B to give the quantum distributed cryptography KEY. The rectified Alice's bits and Bob's bits are compared to give the KEY

ERROR RATE at node B. The time stamp monitor gives the KEY generation time.



BB84 PROTOCOL DESIGN FLOW

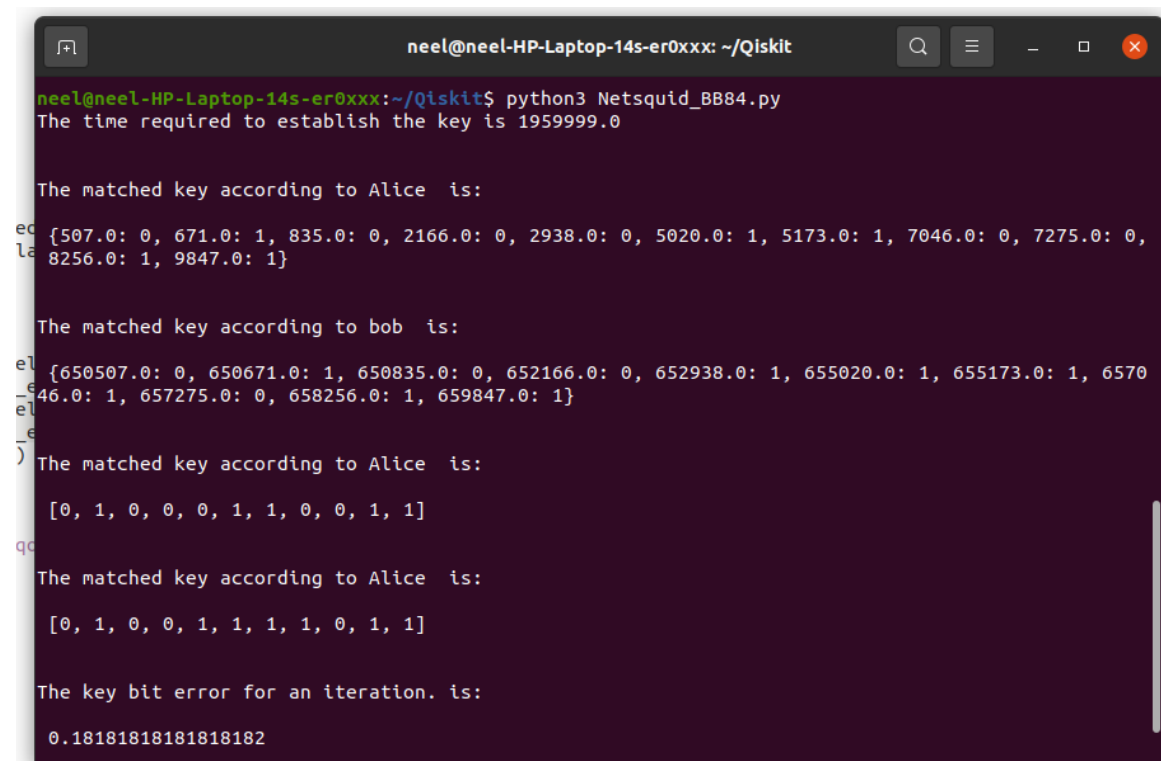
RESULT:

The standard output after simulating the network and carrying out Quantum Key Distribution provides the real world depiction of the BB84 QKD process with practical network delay, channel noise, channel loss on the quantum channel and estimates the delay in generating the KEY. It also gives the error in the keys generated at node A and B due to the above errors.

The suitable standard input for observing output characteristics is : channel length = 100kms, photonic_loss_lenth = 0.2, depolarisation rate = 10^3 . The output for this standard input is given:-

Key bit error rate = 0.18 (as visible, 8 out of 10 bits of the key generated at node A and B match).

Time required to generate the key = 1.959999s.



```
neel@neel-HP-Laptop-14s-er0xxx: ~/Qiskit
neel@neel-HP-Laptop-14s-er0xxx:~/Qiskit$ python3 Netsquid_BB84.py
The time required to establish the key is 1959999.0

The matched key according to Alice is:
{507.0: 0, 671.0: 1, 835.0: 0, 2166.0: 0, 2938.0: 0, 5020.0: 1, 5173.0: 1, 7046.0: 0, 7275.0: 0, 8256.0: 1, 9847.0: 1}

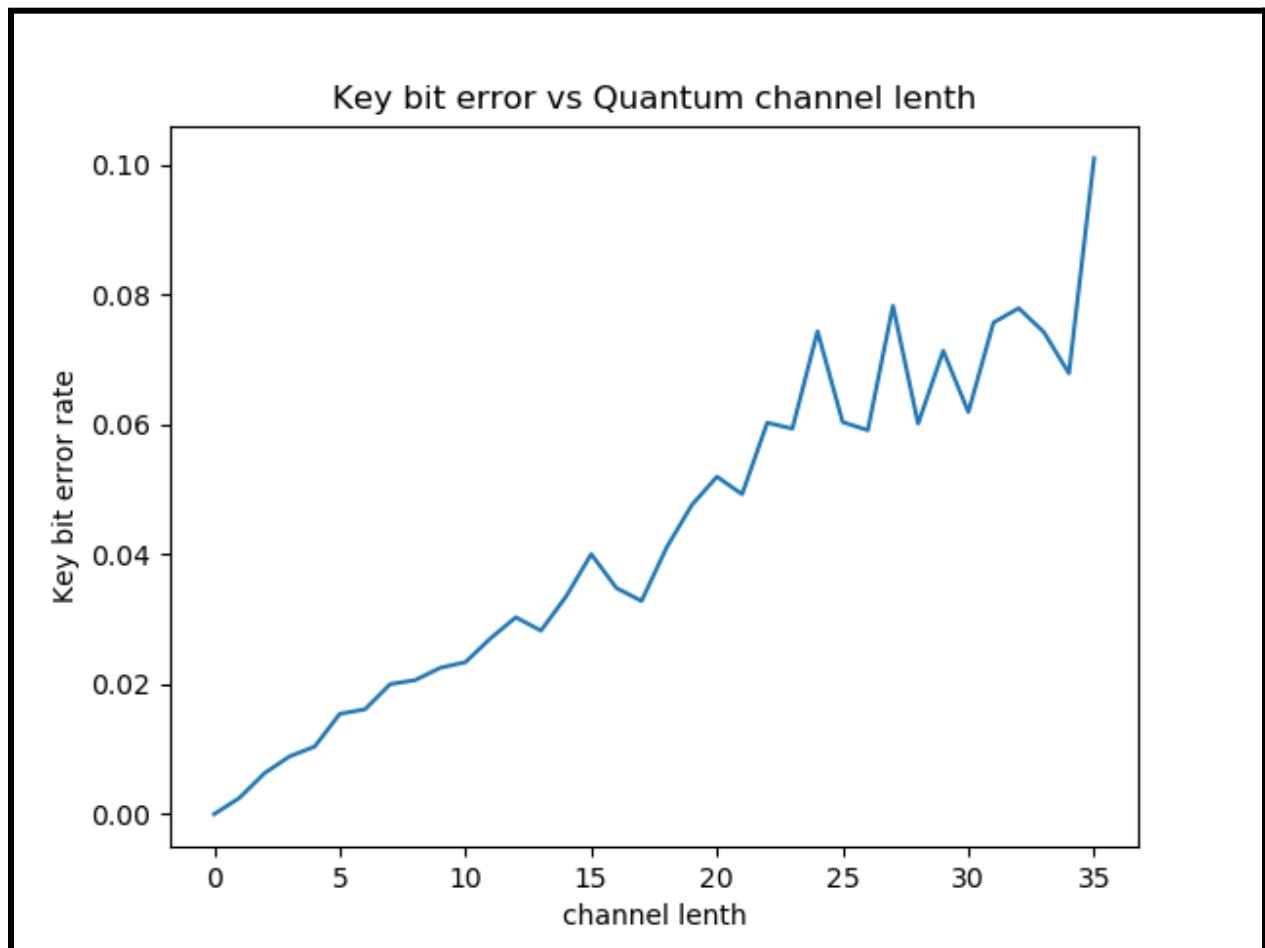
The matched key according to bob is:
{650507.0: 0, 650671.0: 1, 650835.0: 0, 652166.0: 0, 652938.0: 1, 655020.0: 1, 655173.0: 1, 657046.0: 1, 657275.0: 0, 658256.0: 1, 659847.0: 1}

The matched key according to Alice is:
[0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1]

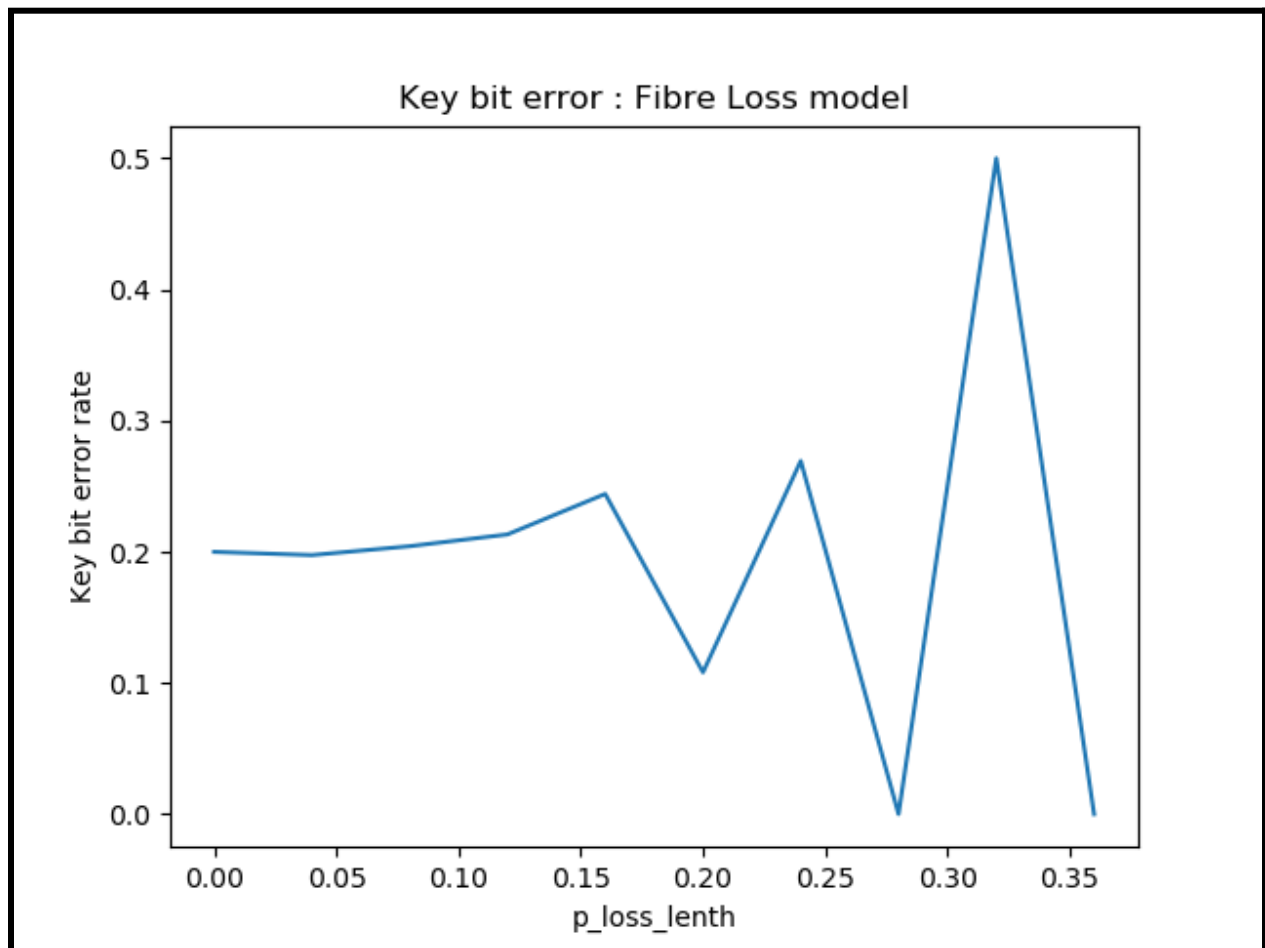
The matched key according to Alice is:
[0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1]

The key bit error for an iteration. is:
0.181818181818182
```

Quantum Delay Model : The key bit error rate with respect to the channel length is given below.



Quantum Fiber Loss Model: The key bit error rate with respect to the photonic loss length is given below.



Quantum Noise Model : The key bit error rate with respect to the depolarisation rate is given below.



INFERENCE :

The Keys generated at node A and B are slightly different from each other due to various quantum channel errors and the accuracy of the keys can be denoted by the KEY BIT ERROR RATE. The time delay to generate the keys varies with different channel specifications and randomly.

The key bit error increases with the channel length linearly.

The key bit error increases slightly with photonic loss length and is smooth before loss length < 0.15 and becomes rougher and more random as loss length tends to 0.4.

The key bit error also increases sharply with the depolarisation rate.

Apart from the above characteristics the key bit error rate also shows properties of gaussian randomness as expected by any quantum particle based system.

Conclusion:

Bit error rate was due to the measurement error in the Quantum Computer causing changes in the bases in the first simulation and due to practical channel errors in the second simulation.

For the Quantum Network, the Bit error rate varied with parameterised models such as in:-

1. Delay Model where the Bit error varied linearly with increase in length of the Quantum Channel.
2. The Key bit error increases smoothly in the beginning till 0.15 photonic loss length and varies abruptly with increased key rate fluctuations as the photonic loss length increases beyond the threshold.
3. Key Bit error increases sharply with depolarization rate.
4. Gaussian Randomness is observed while measurement of the key bases.

This concludes that a Noisy Quantum Computer with Noisy Quantum Channel will have a long time delay with variable Bit error rate.