Codes

quantumMedium.py

```
from flask import Flask, request
from qiskit import *
from tqdm import tqdm

key_length = 2500
qubits = None
comp = Aer.get_backend("qasm_simulator")

def prepare(inp):
    """Prepares the qubits and returns a list of key_length quantum circuits."""
    bits = inp["bits"]
    bases = inp["bases"]
    message = []
    for i in tqdm(range(key_length)):
        qc = QuantumCircuit(1,1)
        if int(bits[i]): qc.x(0)
        if bases[i] == "*": qc.h(0)
        qc.barrier()
        message.append(qc)
    return message

def measure1(qc, basis):
    """Measures the supplied qubit in the desired basis and returns the result."""
    if basis == "*":
        qc.h(0)
    elif basis != "+":
        raise ValueError('"basis" must be "+" or "*"!')
    qc.measure(0, 0)
    results = list(execute(qc, comp, shots = 1).result().get_counts().keys())[0]
    # if random() > 0.85: results = str(1-int(results))  # adding errors
    return results

def measure(qubits, mbasis):
    """takes all the qubits and all the bases and returns a string containing
measurements."""
    if len(qubits) != len(mbasis):
        raise ValueError(f"length of qubits (= {len(qubits)}) and length of mbasis
(= {len(mbasis)}) aren't equal")
    ret = ""
    for i in tqdm(range(len(mbasis))):
        qc = qubits[i]
        basis = mbasis[i]
        ret += measure1(qc, basis)
    return ret

app = Flask(__name__)
```

```
@app.route("/")
def login():
    """Anyone can know the key-length by raising a request to this endpoint."""
    print(f"someone asks for the key-length, Quantum Medium return {key_length}")
    return str(key_length)

@app.route("/send_qubit", methods=["POST"])
def sendqubit():
    """
    Alice will send a dictionary of the form:
    data = {
            "bits": bits,  # key_length long string of 0s and 1s
            "bases": bases,  # key_length long string of + and *
            "name": name  # here name = "Alice"
            }

    this function will generate quantum circuits from then and store them in a
list
    """
    global qubits
    name = request.form["name"]
    if name == "alice":
        print(f"\nAlice sends all the {key_length} qubit")
        qubits = prepare(request.form)
        return "Qubit Added"
    else:
        return f"{name.title()} is not authorised to send qubits!"

@app.route("/read_qubits", methods=["POST"])
def getqubit():
    """anyone can send a request along with a choice of bases at this
    endpoint to get the qubits read in that base."""
    if not qubits:
        print("\nSomeone sends bases and tries to measure the qubits. But Alice
hasn't sent the Qubits yet! So he waits!")
        return "Wait"
    print("\nSomeone measures the qubits in his/her choice of bases.")
    bases = request.form["basis"]
    return measure(qubits, bases)

app.run(
    host = "localhost",  # change this to your local ip
    port = 5050,
    debug = True,
 )
```

classicalMedium.py

```
from flask import Flask, request
```

```
app = Flask(__name__)

# Defining different values
bob_bases, alice_bases = None, None
sample = None
status = None

@app.route('/get_bob_bases', methods=['POST'])
def get_bob_bases():
    """Alice sends her bases and requests for Bob's bases.
    If Bob has already sent his bases, it returns the bases,
    otherwise, it returns "wait", so that Alice waits for
    some time and comes back again to ask for Bob's bases."""
    global alice_bases
    alice_bases = request.form['bases']
    print(f"\nAlice sent her bases and is requesting for Bob's bases.", end = " ")
    if bob_bases:
        print("The Classical Medium already has Bob's bases, so it gives it to
her.")
        return bob_bases
    print("But Bob hasn't sent his bases yet, so she waits.")
    return "wait"

@app.route('/get_alice_bases', methods=['POST'])
def get_alice_bases():
    """Similarly Bob sends his bases and requests for Alice's bases.
    If Bob has already sent his bases, it returns the bases, else returns
"wait"."""
    global bob_bases
    bob_bases = request.form['bases']
    print(f"\nBob sent his bases and is requesting for Alice's bases.", end = " ")
    if alice_bases:
        print("The Classical Medium already has Alice's bases, so it gives it to
him.")
        return alice_bases
    print("But Alice hasn't sent her bases yet, so he waits.")
    return "wait"

@app.route("/send_sample", methods=["POST"])
def send_sample():
    """Alice sends her sample."""
    global sample
    print(f"\nAlice sent her sample.")
    sample = request.form["sample"]
    return "Got it!"

@app.route("/get_sample")
def get_sample():
    """Bob requests for Alice's sample
    If Alice has sent her sample, it returns the sample, else returns "wait"."""
    print(f"\nBob is requesting for Alice's sample.", end = " ")
    if sample:
        print("The Classical Medium already has Alice's sample, so it gives it to
him.")
```

```
            return sample
        print("But Alice hasn't sent her sample yet, so he waits.")
        return "wait"

@app.route("/status", methods=["GET", "POST"])
def get_status():
    """Bob after knowing Alice's sample, calculates the match factor.
    Depending upon the match factor he either decides to keep the key or not.
    He declares his decision in the classical Medium.
    Later Alice requests for the decision."""
    global status
    if request.method == "GET":
        print(f"\nAlice is requesting for Bob's decision on whether to proceed
with the key or start the process of resending another key.")
        if status != None:
            print("The Classical Medium already has Bob's decision, so it gives it
to her.")
            return status
        print("But Bob hasn't sent his decision yet, so she waits.")
        return "wait"
    else:
        print(f"\nBob is sending his decision on whether to proceed with the key
or start the process of resending another key.")
        status = request.form["status"]
        return "done"

app.run(
    host = "localhost",  # change this to your local ip
    port = 5000,
    debug = True,
 )
```

alice.py

```
from random import choice as IDK
from random import sample as take_any
import requests
from time import sleep

name = "alice"
print(f"Hi! This is {name.title()}!")

qIP, qPORT = "localhost", "5050"
cIP, cPORT = "localhost", "5000"
quantumMedium  =  f"http://{qIP}:{qPORT}/"  # URL of quantum channel
classicalMedium = f"http://{cIP}:{cPORT}/"  # URL of classical channel

def prepare(key_length):
    """prepares key_length number of random choices for qubits
    to be prepared in. a denotes the values of the qubits and b
    denotes the corresponding bases to be used to encode the qubit"""
```

```
    a = ""  # X or not
    b = ""  # H or not
    for i in range(key_length):
        a += IDK(["0", "1"])
        b += IDK(["+", "*"])
    return a, b

def sendQubit(key_length):
    """gets the qubit preparation choices and sends them to quantum channel"""
    bits, bases = prepare(key_length)
    requests.post(f"{quantumMedium}send_qubit",
                  data = {
                      "bits": bits,
                      "bases": bases,
                      "name": name
                      }
                  ).text
    return bits, bases

def get_bob_bases(bases):
    try:
        ret = requests.post(f"{classicalMedium}get_bob_bases", data = {"bases":
bases}).text
    except:
        sleep(key_length/120)
        return get_bob_bases(bases)
    if  ret == "wait":
        sleep(key_length/120)
        return get_bob_bases(bases)
    return ret

def send_sample(usable_bits, n = 0.3):
    """prepares and sends a sample of the qubits to quantum channel"""
    sample_indices = sorted(
        take_any(usable_bits, int((key_length/2)*n))
    )
    print("sample length =", len(sample_indices))
    sampleD = {x: bits[x] for x in sample_indices}

    # Encoding it before sending
    sample = ""
    for index, bit in sampleD.items():
        sample += f"{index}:{bit}-"
    sample = sample[:-1]
    # sending
    requests.post(f"{classicalMedium}send_sample", data = {"sample": sample}).text

    return sampleD

def compile_key(bits, sample, usable_bits):
    """compiles the key from the usable bits and discarding the sample"""
    use = [i for i in range(key_length) if i in usable_bits and i not in
sample.keys()]
    ret = ""
```

```
    for i in use:
        ret += bits[i]
    return ret

def successful():
    try:
        ret = requests.get(f"{classicalMedium}status").text
    except:
        sleep(5)
        return successful()
    if  ret == "wait":
        sleep(5)
        return successful()
    if ret == "success":
        return True
    return False



# Know the key length
key_length = int(requests.get(quantumMedium).text)

# Send those many qubit preparation choices to quantum channel.
# Quantum channel will prepare them as requested
bits, alice_bases = sendQubit(key_length)

# Get the bases of Bob
bob_bases = get_bob_bases(alice_bases)

# know which qubits have been read by Bob in the correct basis
# to calculate which qubits are usable and which aren't
usable_bits = [i for i in range(key_length) if bob_bases[i] == alice_bases[i]]

# Send a sample of the usable qubits to quantum channel
sample = send_sample(usable_bits, n = 0.3)

# If successful, compile the key, else abort the mission
if successful():
    key = compile_key(bits, sample, usable_bits)
    print(f"key = {key}")
else:
    print("Eve Detected, mission abort!")
```

bob.py

```
import requests
from random import choice as IDK
from time import sleep

name = "bob"
print(f"Hi! This is {name.title()}!")
```

```
qIP, qPORT = "localhost", "5050"
cIP, cPORT = "localhost", "5000"
quantumMedium  =  f"http://{qIP}:{qPORT}/"  # URL of quantum channel
classicalMedium = f"http://{cIP}:{cPORT}/"  # URL of classical channel

def readQubit():
    bases = ""
    for i in range(key_length):
        bases += IDK(["+", "*"])
    # print(basis)
    bits = requests.post(f"{quantumMedium}read_qubits", data = {"basis":
bases}).text
    if bits == "Wait":
        print("waiting...")
        sleep(1)
        return readQubit()
    return [bits, bases]

def get_alice_bases(bases):
    try:
        ret = requests.post(f"{classicalMedium}get_alice_bases", data = {"bases":
bases}).text
    except:
        sleep(1)
        return get_alice_bases(bases)
    if  ret == "wait":
        sleep(1)
        return get_alice_bases(bases)
    return ret

def get_sample():
    try:
        ret = requests.get(f"{classicalMedium}get_sample").text
    except:
        sleep(1)
        return get_sample()
    if  ret == "wait":
        sleep(1)
        return get_sample()
    return ret

def process_sample(sample):
    ss = sample.split("-")
    ret = {}
    for s in ss:
        a, b = s.split(":")
        ret[int(a)] = int(b)
    return ret

def compile_key(bits, sample, usable_bits):
    use = [i for i in range(key_length) if i in usable_bits and i not in
sample.keys()]
    ret = ""
```

```
    for i in use:
        ret += bits[i]
    return ret

key_length = int(requests.get(quantumMedium).text)

bits, bob_bases = readQubit()

alice_bases = get_alice_bases(bob_bases)


usable_bits = [i for i in range(key_length) if bob_bases[i] == alice_bases[i]]

sample = process_sample(get_sample())

check = [sample[i] == int(bits[i]) for i in sample.keys()]

print(f"match = {sum(check)/len(check)*100}%")

if sum(check)/len(check)>=(0.9-0.05):
    requests.post(f"{classicalMedium}status", data = {"status": "success"})
    print("Success")
    key = compile_key(bits, sample, usable_bits)
    print(f"key length = {len(key)}")
    print(f"key = {key}")
else:
    requests.post(f"{classicalMedium}status", data = {"status": "abort"})
    print("Eve Detected, mission abort!")
```

eve.py

```
import requests
from random import choice as IDK
from time import sleep

name = "eve"
print(f"Hi! This is {name.title()}!")

qIP = "localhost"
cIP = "localhost"
quantumMedium  =  f"http://{qIP}:5050/"  # quantum channel in port 5050
classicalMedium = f"http://{cIP}:5000/"  # classical channel in port 5000

def readQubit():
    bases = ""
    for i in range(key_length):
        bases += IDK(["+", "*"])
    # print(basis)
    bits = requests.post(f"{quantumMedium}read_qubits", data = {"basis":
bases}).text
    if bits == "Wait":
```

```
            print("waiting...")
            sleep(1)
            return readQubit()
        return [bits, bases]

    key_length = int(requests.get(quantumMedium).text)

    bits, bob_bases = readQubit()

    print("completed eavesdropping!")
```