Simulation for Quantum Key Distribution
(research progress)

Takihara Yudai – 3rd year undergraduate student
The University of Aizu

Contents

1. What is the Quantum Key Distribution(QKD)?
2. BB84 Protocol
3. The algorithm and simulation in Python
4. Challenge

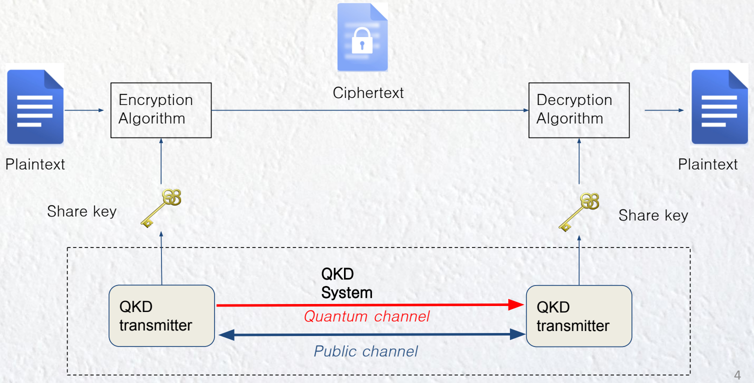
1. What is Quantum Key Distribution(QKD)?

1. What is Quantum Key Distribution(QKD)?

Quantum Key Distribution is a technology that relies on quantum physics to secure the distribution of symmetric encryption keys

- Quantum channel
  - Used for distribution of Qubit.
  - Prevent information leakage and interception

- Public channel
  - share basis
  - announce notification of whether the basis is equal to each other



2. BB84 Protocol

2. BB84 Protocol

- **Operating scheme**
  - Prepare and measure
  - Entanglement-based
- **Implementation**
  - Discrete-variable
  - Continuous-variable
  - Non-coherent CV

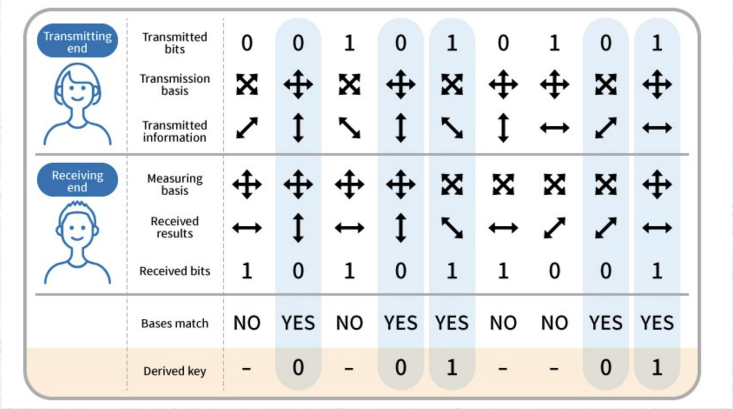
BB84

The popular protocol for Quantum Key Distribution

This protocol is named after the initials of the two developers and the year this protocol was published

2. BB84 Protocol

Assume that sender and receiver generate a key for communication.



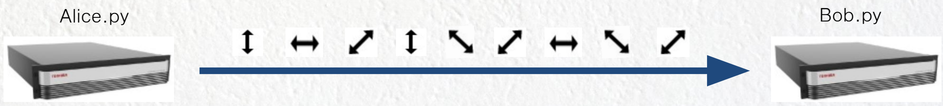
Transmitting end	Transmitted bits	0	0	1	0	1	0	1	0	1
	Transmission basis	✕	✛	✕	✛	✕	✛	✛	✕	✛
	Transmitted information	⤢	↕	⤡	↕	⤡	↕	↔	⤢	↔
Receiving end	Measuring basis	✛	✛	✛	✛	✕	✕	✕	✕	✛
	Received results	↔	↕	↔	↕	⤡	↔	⤢	⤢	↔
	Received bits	1	0	1	0	1	1	0	0	1
	Bases match	NO	YES	NO	YES	YES	NO	NO	YES	YES
	Derived key	-	0	-	0	1	-	-	0	1

https://www.global.toshiba/ww/company/digitalsolution/articles/tsoul/38/004.html

3. Simulation of BB84

3. Simulation of BB84

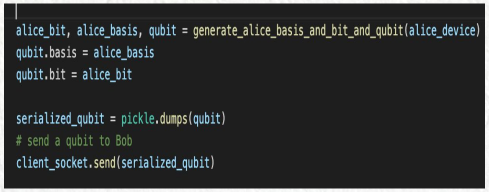
First, Alice generates randomly basis and bit value, and decides Qubit state, then she send the qubit to Bob.



basis \ bit value		0	1
Rectilinear basis (0)	✥	↕ [1, 0] (default)	↔ [0, 0.7071067811865475+0j]
Diagonal basis (1)	✖	⤢ [0.7071067811865475+0j, 0.7071067811865475+0j]	⤡ [0.4999999999999999+0j, −0.4999999999999999+0j]

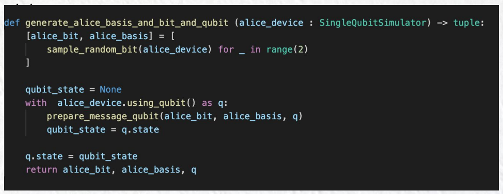
3. Simulation of BB84

Alice.py



```
alice_bit, alice_basis, qubit = generate_alice_basis_and_bit_and_qubit(alice_device)
qubit.basis = alice_basis
qubit.bit = alice_bit

serialized_qubit = pickle.dumps(qubit)
# send a qubit to Bob
client_socket.send(serialized_qubit)
```

generate_alice_basis_and_bit_and_qu



```
def generate_alice_basis_and_bit_and_qubit (alice_device : SingleQubitSimulator) -> tuple:
    [alice_bit, alice_basis] = [
        sample_random_bit(alice_device) for _ in range(2)
    ]

    qubit_state = None
    with  alice_device.using_qubit() as q:
        prepare_message_qubit(alice_bit, alice_basis, q)
        qubit_state = q.state

    q.state = qubit_state
    return alice_bit, alice_basis, q
```

In "*generate_alice_basis_and_bit_and_qubit()*", the state of the qubit is changed by H− and X−operations, depending on the value of the generated bit and basis (default value is [1, 0]).

3. Simulation of BB84

Second, Bob generates randomly basis and he decides his bit based on his basis and the qubit state which received from Alice in step 1.

measuring basis	✥	✥	✥	✥	✖	✖	✖	✖	✥
Qubit state	↔	↕	↔	↕	⤡	↔	⤢	⤢	↔
decided Bob's bit	1	0	1	0	1	1	0	0	1

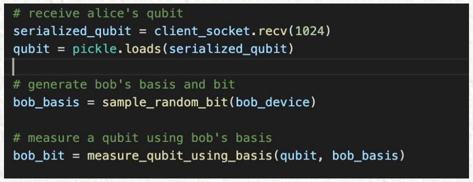
3. Simulation of BB84

Second, Bob generates randomly basis and he decides his bit based on his basis and the qubit state which received from Alice in step 1.

measuring basis	✥	✥	✥	✥	✖	✖	✖	✖	✥
Qubit state	↔	↕	↔	↕	⤡	↔	⤢	⤢	↔
decided Bob's bit	1	0	1	0	1	1	0	0	1

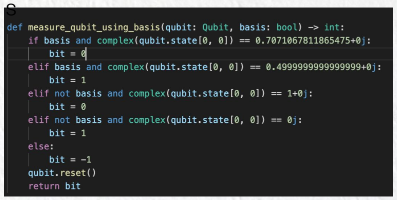
3. Simulation of BB84

Bob.py



```
# receive alice's qubit
serialized_qubit = client_socket.recv(1024)
qubit = pickle.loads(serialized_qubit)

# generate bob's basis and bit
bob_basis = sample_random_bit(bob_device)

# measure a qubit using bob's basis
bob_bit = measure_qubit_using_basis(qubit, bob_basis)
```

measure_qubit_using_basis

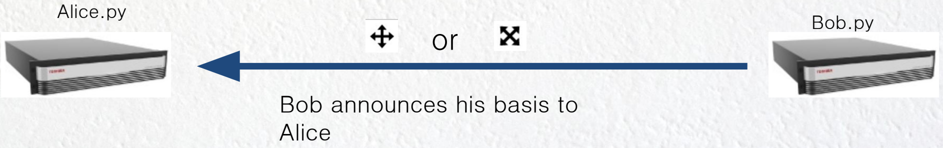


```
def measure_qubit_using_basis(qubit: Qubit, basis: bool) -> int:
    if basis and complex(qubit.state[0, 0]) == 0.7071067811865475+0j:
        bit = 0
    elif basis and complex(qubit.state[0, 0]) == 0.4999999999999999+0j:
        bit = 1
    elif not basis and complex(qubit.state[0, 0]) == 1+0j:
        bit = 0
    elif not basis and complex(qubit.state[0, 0]) == 0j:
        bit = 1
    else:
        bit = -1
    qubit.reset()
    return bit
```

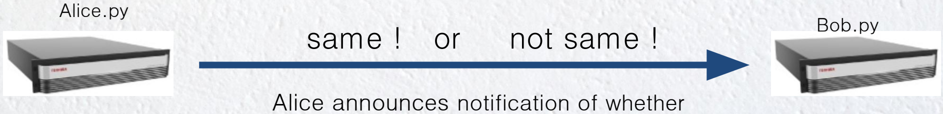
In “measure_qubit_using_basis()”, Bob measure Alice’s qubit using Bob’basis. In concrete terms, the combination of the state of the qubit and the value of the basis (0 or 1) determines the bits of the bob.

3. Simulation of BB84

Next, Bob announces his basis to Alice. After that, Alice compares her basis to his basis. She announces notification of whether the basis is equal to each other. If it is the same, add bit value to key but if not, the bit value is discarded each other.



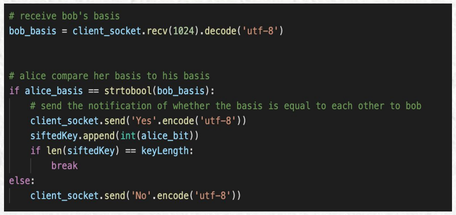
Bob announces his basis to Alice



Alice announces notification of whether the basis is equal to each other

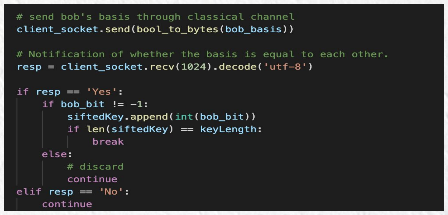
3. Simulation of BB84

Alice.py



```
# receive bob's basis
bob_basis = client_socket.recv(1024).decode('utf-8')


# alice compare her basis to his basis
if alice_basis == strtobool(bob_basis):
    # send the notification of whether the basis is equal to each other to bob
    client_socket.send('Yes'.encode('utf-8'))
    siftedKey.append(int(alice_bit))
    if len(siftedKey) == keyLength:
        break
else:
    client_socket.send('No'.encode('utf-8'))
```

Bob.py

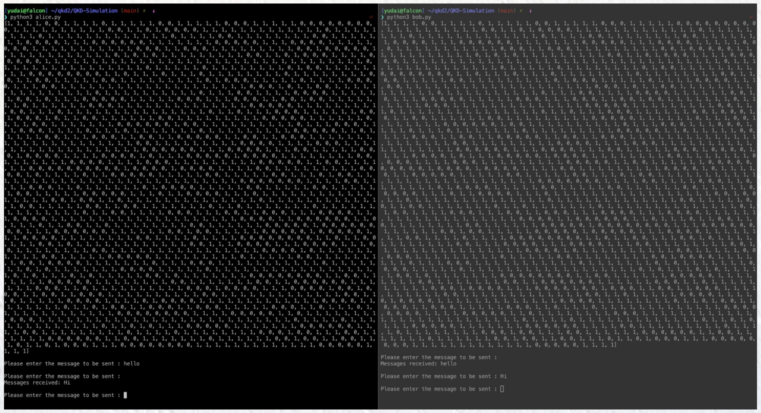


```
# send bob's basis through classical channel
client_socket.send(bool_to_bytes(bob_basis))

# Notification of whether the basis is equal to each other.
resp = client_socket.recv(1024).decode('utf-8')

if resp == 'Yes':
    if bob_bit != -1:
        siftedKey.append(int(bob_bit))
        if len(siftedKey) == keyLength:
            break
    else:
        # discard
        continue
elif resp == 'No':
    continue
```

After Bob sends his basis, Alice receives the bases from Bob So she compares each base with an if construct. She then tells Bob the result of the comparison (Yes or No). If Yes, a bit value is added as part of the key, but each bits is discarded if not.

3. Simulation of BB84

Result of key generation up to this step.



```
[yudai@falcon] ~/qkd2/QKD-Simulation (main) ⚡ ↓
❯ python3 alice.py
[1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1
[illegible]
1, 1, 1]

Please enter the message to be sent : hello

Please enter the message to be sent :
Messages received: Hi

Please enter the message to be sent : █
```

```
[yudai@falcon] ~/qkd2/QKD-Simulation (main) ⚡ ↓
❯ python3 bob.py
[1, 1, 1, 1,
[illegible]
 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1]

Please enter the message to be sent :
Messages received: hello

Please enter the message to be sent : Hi

Please enter the message to be sent : ▯
```

4. Challenge

4. Challenge

The rest of the post-processing

1 Parameter estimation : The procedure that Alice and Bob want to compute a guess for the error rate in the quantum channel.

2 Error correction : The procedure that Alice and Bob perform certain steps to correct errors in their keys and increase the secrecy of their key.

3 Privacy amplification : The procedure that minimizes Eve’s knowledge of the key

Thank for listening