Using Qiskit to create Quantum Games

myron.giannakis@gmail.com
/myron-giannakis/

Myron Giannakis

Computer Engineering student, coordinator of Quantum Computing students' team

Contents

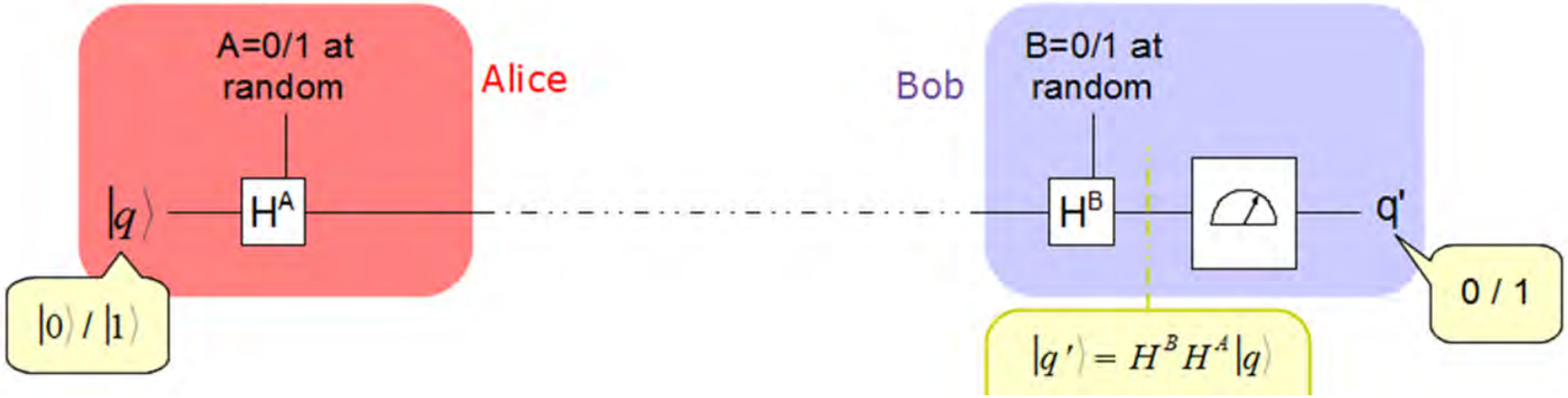
- Who am I
- Quantum Key Distribution (QKD)
- The BB84 Protocol

- The main idea
- The code
- The game

Quantum Key Distribution (QKD)

- BB84 protocol: C. H. Bennett, G. Brassar. "Quantum cryptography: Public key distribution and coin tossing". International Conference on Computers, Systems & Signal Processing. Volume 1. Dec 1984.
- 2 channels, a quantum and an authenticated classical
- No-cloning theorem
- Unbreakable

BB84 Protocol



QUANTUM TRANSMISSION											
Alice's random bits	0	1	1	1	0	0	0	1	1	0	1
Random sending bases	D	D	R	R	R	R	D	R	D	D	R
Random receiving bases	R	D	R	D	D	R	R	D	D	D	R
Bits as received by Bob	1	1	1	0	0	0	1	1	1	0	1
PUBLIC DISCUSSION											
Bob reports bases of received bits	R	D	R	D	D	R	R	D	D	D	R
Alice says which bases were correct		OK	OK			OK			OK	OK	OK
Presumably shared information (if no eavesdrop)		1	1			0			1	0	1

The main idea

- Qiskit: IBM's open-source SDK for working with quantum computers
- Differences from the BB84 protocol:
  - Not a bit-string but a character-string, each encoded to a 7-qubit circuit
  - Not key distribution but message sending

```
class CryptoCircuit(QuantumCircuit):···

class ProtocolBB84:
    def sender(message:str):···

    def receiver(circuits:List[CryptoCircuit]):···

    def classicalChannel(sender_gates, receiver_gates):···

def main():···
```

The code – main

```python
def main():
    message = "Hello, World! I am Myron Giannakis.."
    received_message = ""
    final_message = RETAINED_CHAR *len(message)

    # Loop until the message contains only the retained character
    while message.strip(RETAINED_CHAR) != '':
        # Sender
        circuits, sender_gates = ProtocolBB84.sender(message)
        # Receiver
        received_message, receiver_gates = ProtocolBB84.receiver(circuits)
        # Public discussion
        correct_chars = ProtocolBB84.classicalChannel(sender_gates, receiver_gates)

        # Process data
        received_correctly = ''.join([letter if i in correct_chars else RETAINED_CHAR
                                      for i,letter in enumerate(received_message)])

        message = ''.join([ letter if letter!=received_correctly[i] else RETAINED_CHAR
                            for i, letter in enumerate(message) ])

        final_message = ''.join([ letter if letter!=RETAINED_CHAR else received_correctly[i]
                                  for i, letter in enumerate(final_message) ])
```

The code – protocol methods

```python
class ProtocolBB84:
    def sender(message:str):
        circuits, gates = [], []
        for letter in message:

            # Get letter's binary code
            letter = bin(ord(letter))[2:]

            # Create Quantum Circuit
            circuit = CryptoCircuit()

            # Alice's byte
            circuit.initialize(letter)

            # Choose a random sending basis
            gates.append(getrandbits(1))
            circuit.add_gate(gates[-1])

            # Save circuit and gate
            circuits.append(circuit)

        return circuits, gates
```

```python
def receiver(circuits:List[CryptoCircuit]):
    letters, gates = [], []
    for circuit in circuits:

        # Choose a random receiving basis
        gates.append(getrandbits(1))
        circuit.add_gate(gates[-1])

        # Measure the qubits to get the ASCII code
        letter = circuit.get_measurements()
        # Get letter from binary code
        letters.append(chr(int(letter,2)))

    return letters, gates
```

```python
def classicalChannel(sender_gates, receiver_gates):
    correct_chars = []
    for i, (sent,received) in enumerate(zip(sender_gates, receiver_gates)):
        if sent == received :
            correct_chars.append(i)
    return correct_chars
```

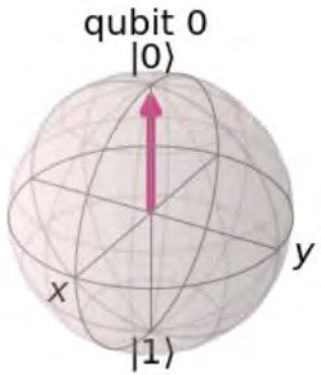
The code – circuit class

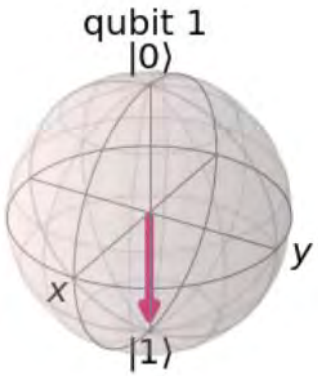
```
class CryptoCircuit(QuantumCircuit):
    def __init__(self, size=7):
        # Each qubit representing a bit in ASCII (7-bit)
        super().__init__(size)

    def initialize(self, code:str):
        code = code[::-1] # reverse order
        for i in range(len(code)):
            self.x(i) if code[i]!='0' else self.i(i)

    def add_gate(self, gate:bool):
        self.h(self.qubits) if gate else self.i(self.qubits)

    def get_measurements(self):
        self.measure_all()
        sim = Aer.get_backend("qasm_simulator")
        counts = execute(self, sim, shots=1).result().get_counts()
        return list(counts)[0]

    def visualize(self):
        sim = Aer.get_backend("statevector_simulator")
        statevector = execute(self, sim, shots=1).result().get_statevector()
        display(plot_bloch_multivector(statevector))
```

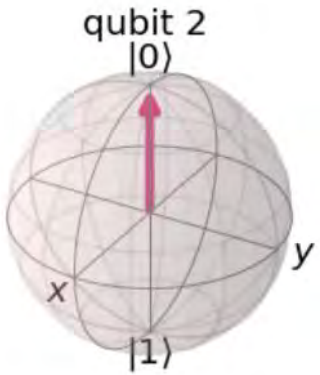
The code - visualizations

```
circ = CryptoCircuit(3)
circ.initialize('010')
circ.visualize()
```

✓ 0.3s

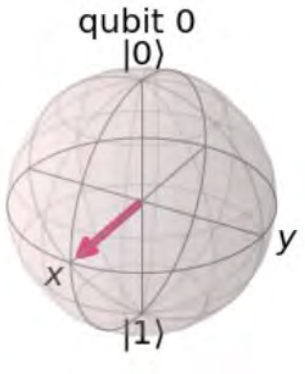


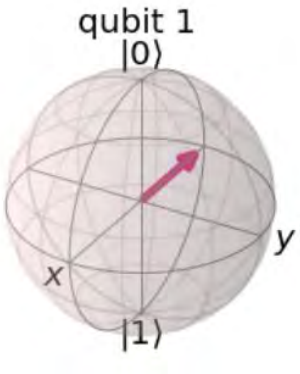


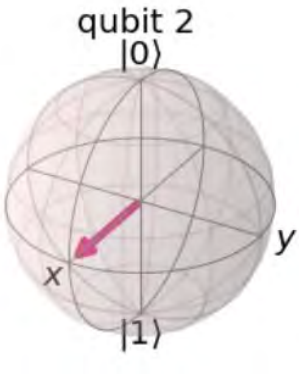



```
circ = CryptoCircuit(3)
circ.initialize('010')
circ.add_gate(1) # apply an H gate
circ.visualize()
circ.get_measurements()
```

✓ 0.4s








```
'001'
```

The game - sender

- Code used by the facilitator to encrypt the message
- Save the circuits in the quantum_channel file and the gates in the classical_channel.

```
message = "BB84 is a Quantum Key Distribution protocol relying on no-cloning theorem."

circuits_out, sender_gates = ProtocolBB84.sender(message)

# Send - write to file
with open("quantum_channel.txt", "bw") as f:
    for circuit in circuits_out:
        f.write( pickle.dumps(circuit) + b'\n\n')

with open("classical_channel.txt", "w") as f:
    for gate in sender_gates:
        f.write( str(gate) )
```

The game - receiver

- No main function and sender method
- receiver and classicalChannel methods to be implemented

```python
class ProtocolBB84:

    def receiver(circuits:list[CryptoCircuit]) -> tuple[list[str], list[int]]:
        letters, gates = [], []

        # TODO

        return letters, gates


    def classicalChannel(sender_gates, receiver_gates):
        correct_chars = []

        # TODO

        return correct_chars
```

Thank you!

Take a look at the GitHub repository for this project:

https://github.com/IEEE-SB-UPatras-Quantum-Computing/Quantum-Cryptography-Game.git