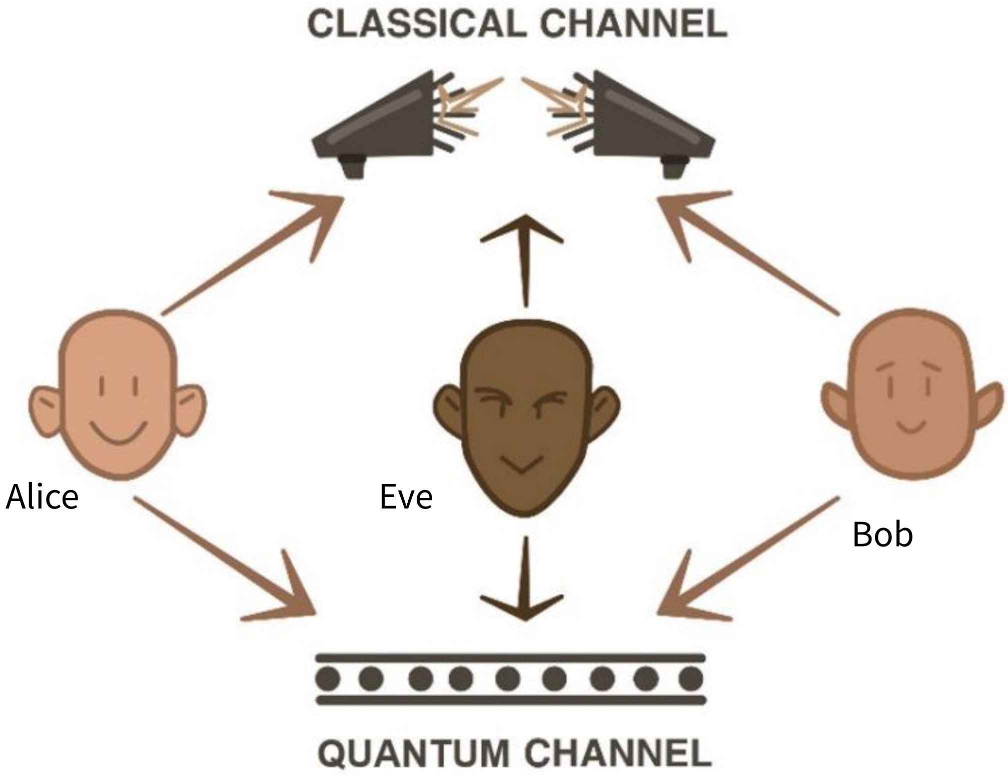
Quantum Key Distribution

Alexandra White

Quantum Key Distribution



What I am trying to solve

- Step 1

Alice chooses a string of random bits, e.g.:

`1000101011010100`

And a random choice of basis for each bit:

`ZZXZXXXZXZXXXXXX`

Alice keeps these two pieces of information private to herself.

- Step 2

Alice then encodes each bit onto a string of qubits using the basis she chose; this means each qubit is in one of the states $|0\rangle$, $|1\rangle$, $|+\rangle$ or $|-\rangle$, chosen at random. In this case, the string of qubits would look like this:

$$|1\rangle|0\rangle|+\rangle|0\rangle|-\rangle|+\rangle|-\rangle|0\rangle|-\rangle|1\rangle|+\rangle|-\rangle|+\rangle|-\rangle|+\rangle|+\rangle$$

This is the message she sends to Bob.

- Step 3

Bob then measures each qubit at random, for example, he might use the bases:

`XZZZXZXZXZXZZZXZ`

And Bob keeps the measurement results private.

- Step 4

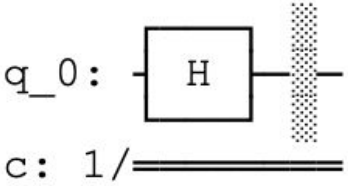
Bob and Alice then publicly share which basis they used for each qubit. If Bob measured a qubit in the same basis Alice prepared it in, they use this to form part of their shared secret key, otherwise they discard the information for that bit.

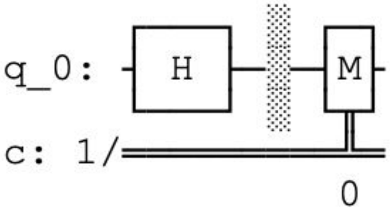
- Step 5

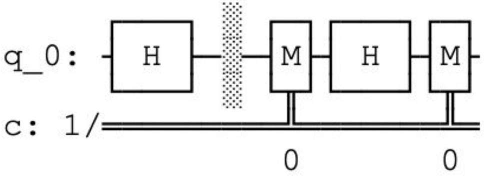
Finally, Bob and Alice share a random sample of their keys, and if the samples match, they can be sure (to a small margin of error) that their transmission is successful.

Suppose Alice sends Bob a qubit, and an eavesdropper (Eve) tries to measure it before it reaches Bob. There is a possibility that Eve's measurement will alter the qubit's state. This may result in Bob receiving the qubit in a different state Alice sent.

Example







In [47]:
```
# import libraries

from qiskit import QuantumCircuit, Aer, transpile, assemble
from qiskit.visualization import plot_histogram, plot_bloch_multivector
from numpy.random import randint
import numpy as np
print("Imports Successful")
```

```
Imports Successful
```

In [48]:
```
np.random.seed(seed=3)
n = 100
#Step 1 is for Alice to generate her random set of bits
alice_bits = randint(2, size=n)# alice_bits is only known to Alice
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]
```

Now we are creating an aray to tell us which
qubits are encoded in which bases. In this case 0 means its prepared
in the Z bases and 1 means its prepared in the X bases

In [49]:
```
alice_bases = randint(2, size=n) #alice_bases is only known to alice
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]
```

In [50]:
```
def encode_message(bits, bases):
    message = []
    for i in range(n):
        qc = QuantumCircuit(1,1)
        if bases[i] == 0: # Preparing the qubit in Z-basis
            if bits[i] == 0:
                pass
            else:
                qc.x(0)
        else: # Preparing the qubit in X-basis
            if bits[i] == 0:
                qc.h(0)
            else:
                qc.x(0)
                qc.h(0)
        qc.barrier()
        message.append(qc)
    return message
```

[51]:
```
message= encode_message(alice_bits, alice_bases)
```

[52]:
```
eve_bases = randint(2, size=n)
intercepted_message = measure_message(message, eve_bases)
print(intercepted_message)
```

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

[53]:
```
#Checking first bit in 'alice_bits'
print('bit = %i' % alice_bits[0])
print('basis = %i' % alice_bases[0])
```

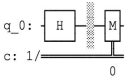
```
bit = 0
basis = 1
```

We see that the first bit in 'alices_bits'is 0 and the bases is X bases
Now we can print the circuit to verify her qubit state

In [54]:
```
message[0].draw() #As seen Alice prepared her qubit in state |+>

#The message of qubits is sent over to bob over Eves quantum channel
```

Out[54]:

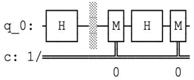


In [56]:
```python
def measure_message(message, bases):
    backend = Aer.get_backend('aer_simulator')
    measurements = []
    for q in range(n):
        if bases[q] == 0: # measuring in Z-basis
            message[q].measure(0,0)
        if bases[q] == 1: # measuring in X-basis
            message[q].h(0)
            message[q].measure(0,0)
        aer_sim = Aer.get_backend('aer_simulator')
        qobj = assemble(message[q], shots=1, memory=True)
        result = aer_sim.run(qobj).result()
        measured_bit = int(result.get_memory()[0])
        measurements.append(measured_bit)
    return measurements
```

In [57]:
```python
bob_results = measure_message(message, bob_bases)

#In the circuit (representing the 0th qubit) it has a X measurment added to it by
message[0].draw()
```

Out[57]:



Alice shows which qubits were encoded in which basis (through eves channel)

The Bob reveals which basis he measured each qubit in

* If bob measured a bit in the same basis Alice prepared the bit in the bit can be key. If they measured the bit in different bases they throw that entry away.

In [58]:
```python
#Creating a function to throw away the non corresponding entry
def remove_garbage(a_bases, b_bases, bits):
    good_bits = []
    for q in range(n):
        if a_bases[q] == b_bases[q]:
            # If both used the same basis, add
            # this to the list of 'good' bits
            good_bits.append(bits[q])
    return good_bits
```

After Alice and Bob discard the useless bits they use the remaining bits to form t

In [59]:
```python
alice_key = remove_garbage(alice_bases, bob_bases, alice_bits)
bob_key = remove_garbage(alice_bases, bob_bases, bob_results)
```

Now Alice and Bob compare a random selection of the bits in their keys to make sure the protocol worked correctly

In [60]:
```python
def sample_bits(bits, selection):
    sample = []
    for i in selection:
        # use np.mod to make sure the
        # bit we sample is always in
        # the list range
        i = np.mod(i, len(bits))
        # pop(i) removes the element of the
        # list at index 'i'
        sample.append(bits.pop(i))
    return sample
```

In [61]:
```python
#Alice and bob both show these publicly, and then remove them from their
#keys because they are no longer secret
sample_size = 15
bit_selection = randint(n, size=sample_size)

bob_sample = sample_bits(bob_key, bit_selection)
print("  bob_sample = " + str(bob_sample))
alice_sample = sample_bits(alice_key, bit_selection)
print("alice_sample = "+ str(alice_sample))
```

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

In [62]:
```python
bob_sample == alice_sample
```

Out[62]: False

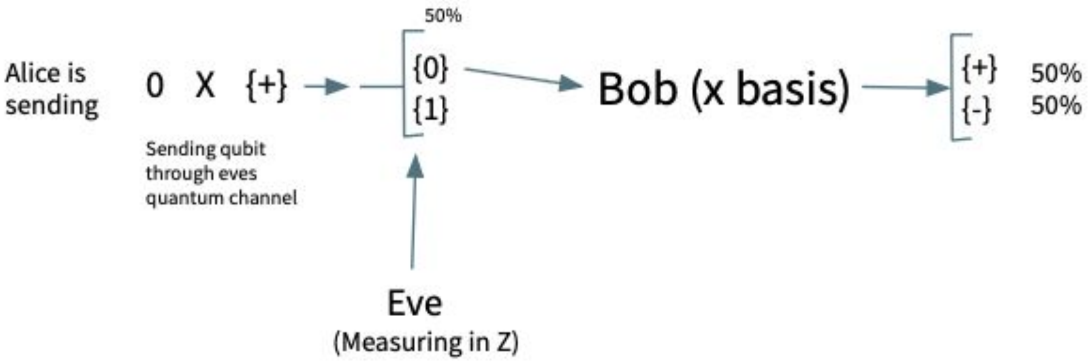
Bob's key and Alice's key do not match. We know this is because Eve tried to read the message between steps 2 and 3, and changed the qubits' states. For all Alice and Bob know, this could be due to noise in the channel, but either way they must throw away all their results and try again- Eve's interception attempt has failed.

If the protocol worked correctly without interference their samples should match

If their samples match, it means (with high probability) alice_key == bob_key. They now share a secret key they can use to encrypt their messages!

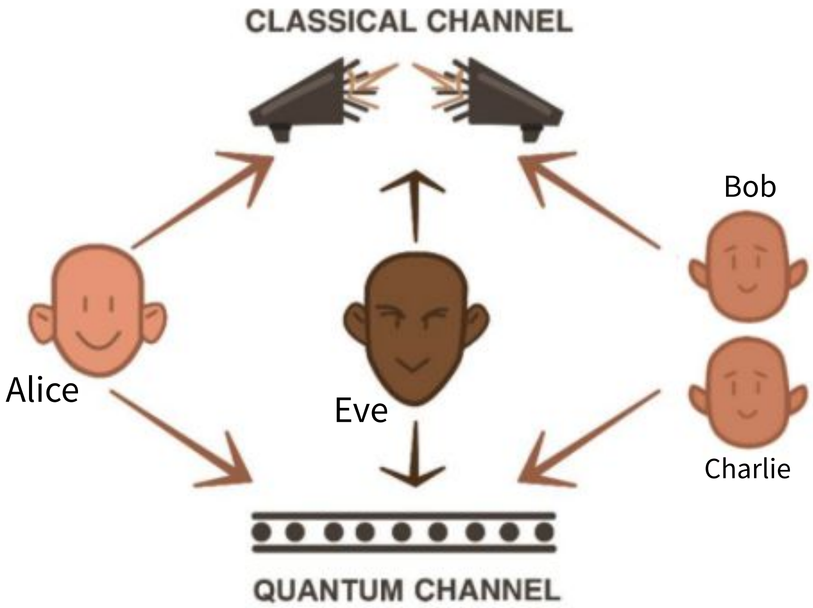
Example Drawn out QKD with interception

message: 0101110
Basis: XZXZZXZ



	0	1
X basis:	{-}	{+}
Z basis:	{0}	{1 }

Multi-User Quantum Key Distribution



How it works

- **Step 1**

Alice chooses a string of random bits, e.g.:

`1000101011010100`

And a random choice of basis for each bit:

`ZZXZXXXZXZXXXXXX`

Alice keeps these two pieces of information private to herself.

- **Step 2**

Alice then encodes each bit onto a string of qubits using the basis she chose; this means each qubit is in one of the states $|0\rangle$, $|1\rangle$, $|+\rangle$ or $|-\rangle$, chosen at random. In this case, the string of qubits would look like this:

$$|1\rangle|0\rangle|+\rangle|0\rangle|-\rangle|+\rangle|-\rangle|0\rangle|-\rangle|1\rangle|+\rangle|-\rangle|+\rangle|-\rangle|+\rangle|+\rangle$$

This is the message she sends to Bob.

- **Step 3**

Bob then measures each qubit at random, for example, he might use the bases:

`XZZZXZXZXZXZZZXZ`

And Bob keeps the measurement results private.

- **Step 4**

Bob and Alice then publicly share which basis they used for each qubit. If Bob measured a qubit in the same basis Alice prepared it in, they use this to form part of their shared secret key, otherwise they discard the information for that bit.

- **Step 5**

Finally, Bob and Alice share a random sample of their keys, and if the samples match, they can be sure (to a small margin of error) that their transmission is successful.

Before Alice encodes each bit onto a string of qubits, She prepares two entangled qubits and sends one of them to charlie and the other one to bob.

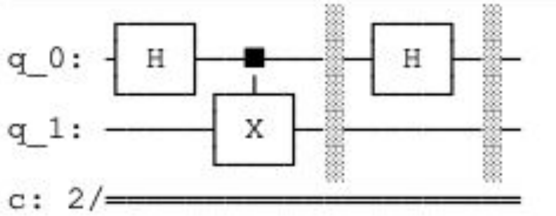
E.g. Alice is sending $|00\rangle$

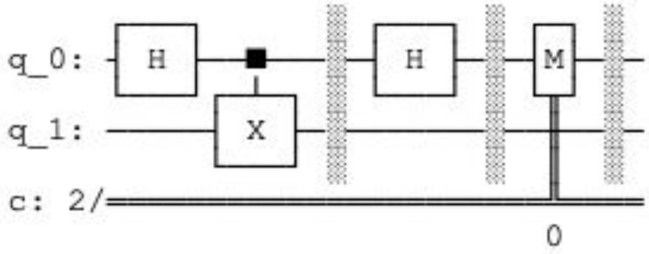
q1 q2

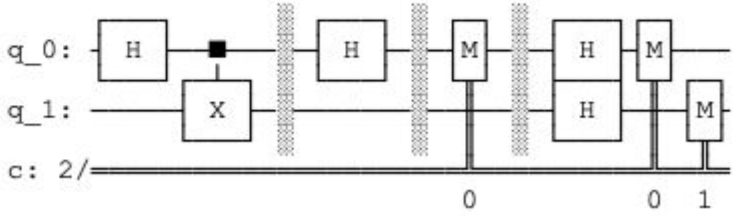
charlie Bob

Alice must entangle everyone's qubit to make sure everyone is measuring the same thing if their basis are the same. You don't need to entangle for single user because there is only one qubit for each message bit

Example







In [204]:
```
np.random.seed(seed=3)
n = 100
#Step 1 is for Alice to generate her random set of bits
alice_bits = randint(2, size=n)# alice_bits is only known to Alice
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]
```

In [205]:
```
alice_bases = randint(2, size=n)#alice_bases is only known to alice
print(alice_bases)
```

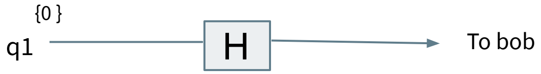
```
[1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 0 0 1 1 0 0 1 1 1 1 0 0
 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 1 1 0 0 1 0 0 1 1 1 1
 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 0 0 0 1 0 1 1 1]
```

In [206]:
```
m = 2
def encode_message(bits, bases):
    message = []
    for i in range(n):
        qc = QuantumCircuit(m, m)
        # entangle qubits
        qc.h(0)
        for k in range(0, m-1):
            qc.cx(k, k+1)
        qc.barrier()
        # encode
        if bases[i] == 0: # Preparing the qubit in Z-basis
            if bits[i] == 0:
                pass
            else:
                qc.x(0)
        else: # Preparing the qubit in X-basis
            if bits[i] == 0:
                qc.h(0)
            else:
                qc.x(0)
                qc.h(0)
        qc.barrier()
        message.append(qc)
    return message
```

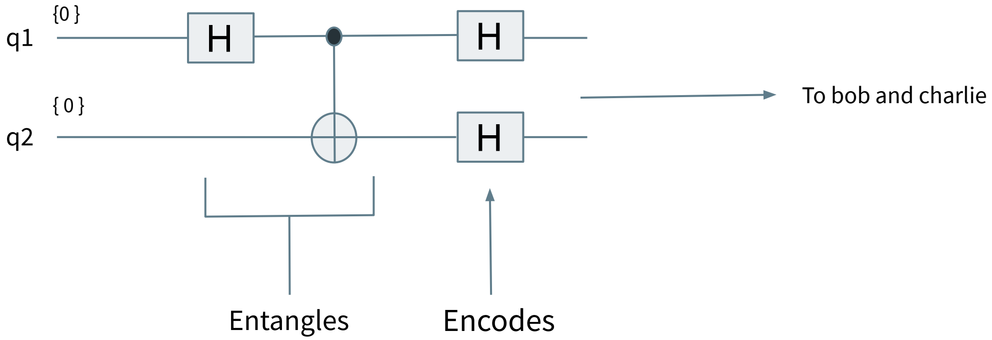
In [207]:
```
message= encode_message(alice_bits, alice_bases)
```

Conclusion

Single receiver



Two-users (receivers)



Sources

https://advances.sciencemag.org/content/advances/7/23/eabe0395.full.pdf
https://qiskit.org/textbook/ch-algorithms/quantum-key-distribution.html#4.-Qiskit-Example:-With-Interception
https://en.wikipedia.org/wiki/Quantum_key_distribution
https://qiskit.org/textbook/ch-states/atoms-computation.html#4.1-Encoding-an-input-