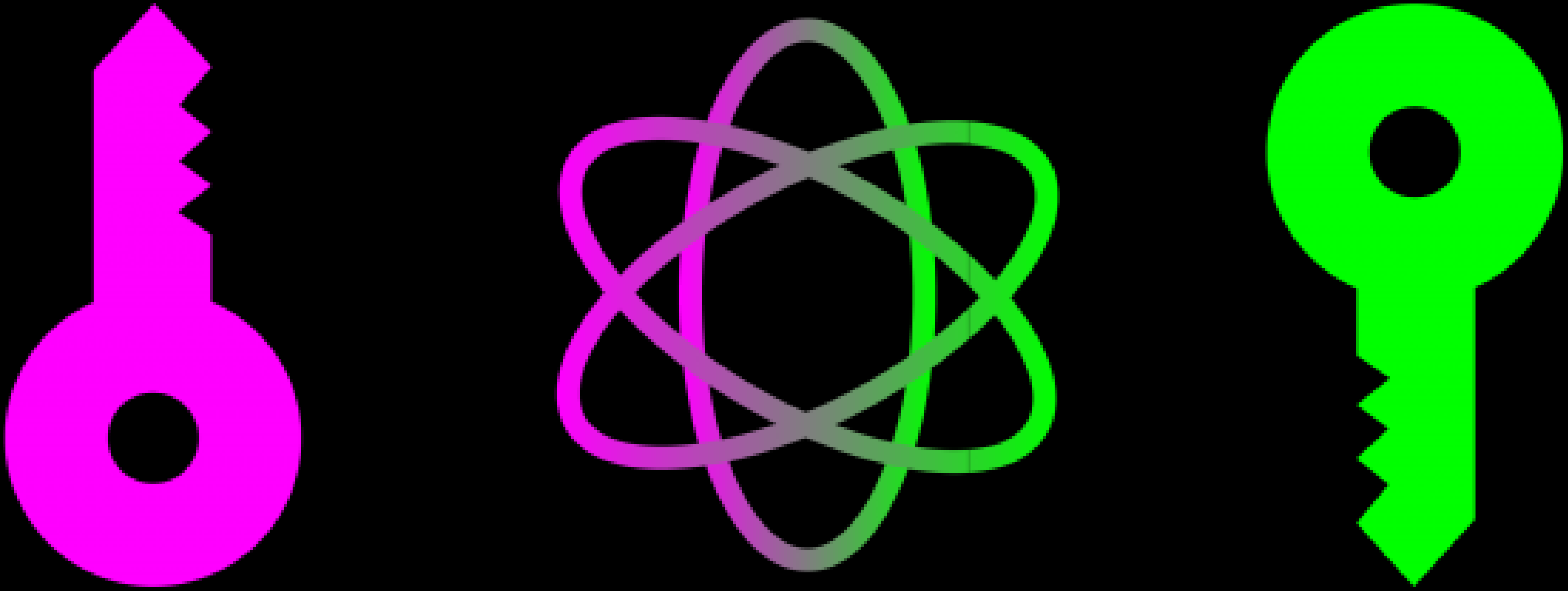
Project: Solving problems on quantum hardware WS2020/21



A Quantum Key Distribution Protocol

By Daniel Fan and Akash Vani

09 October 2020

Image credits: technologyreview.com

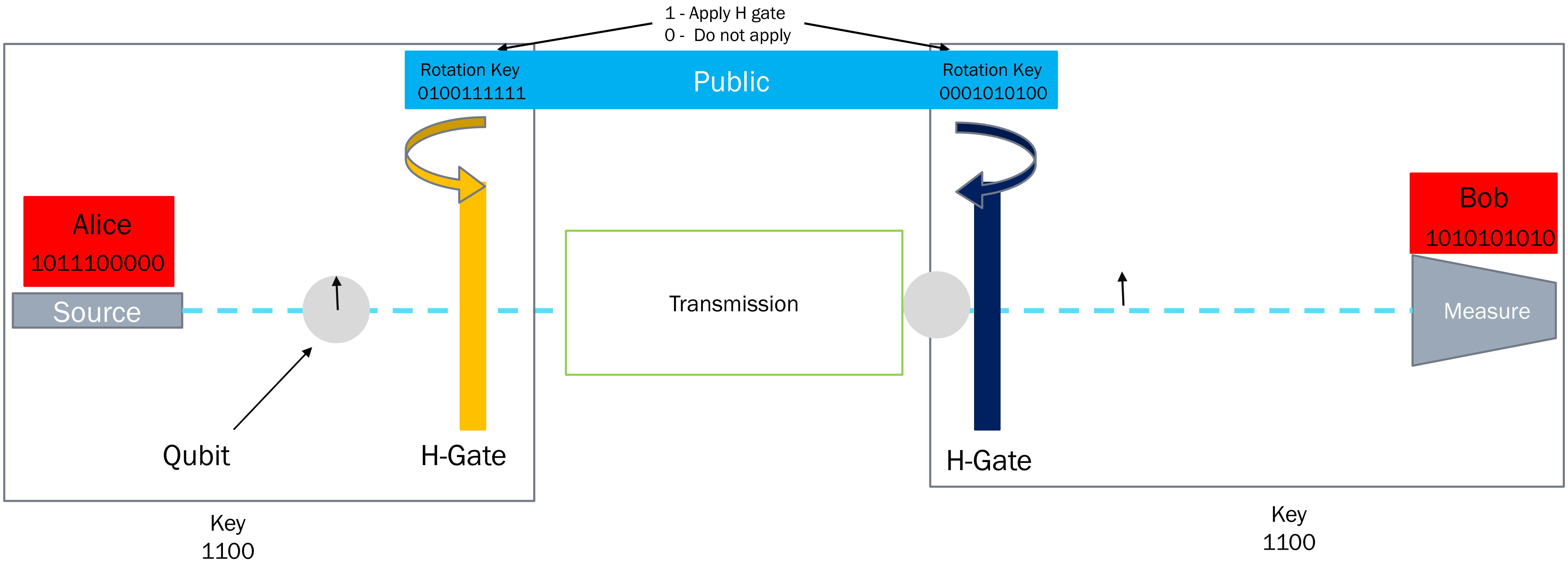
Aim of the project

To show the working of Quantum Key Distribution (QKD) using a variation of a protocol developed in 1984.

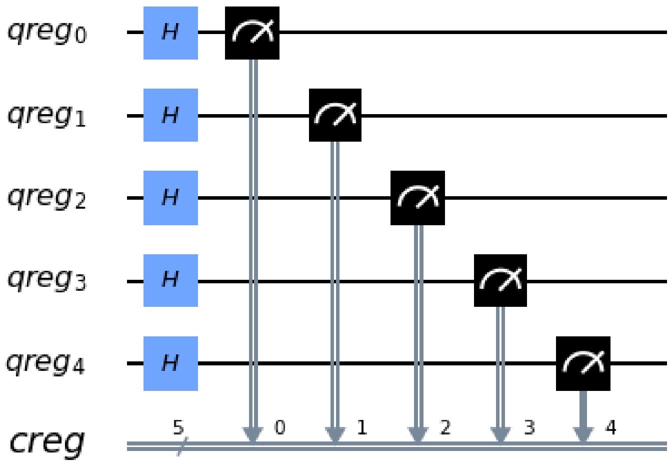
We show, how Alice creates a qubit in superposition and sends it to Bob, who measures this superposition to generate a key.

Images: PNGegg

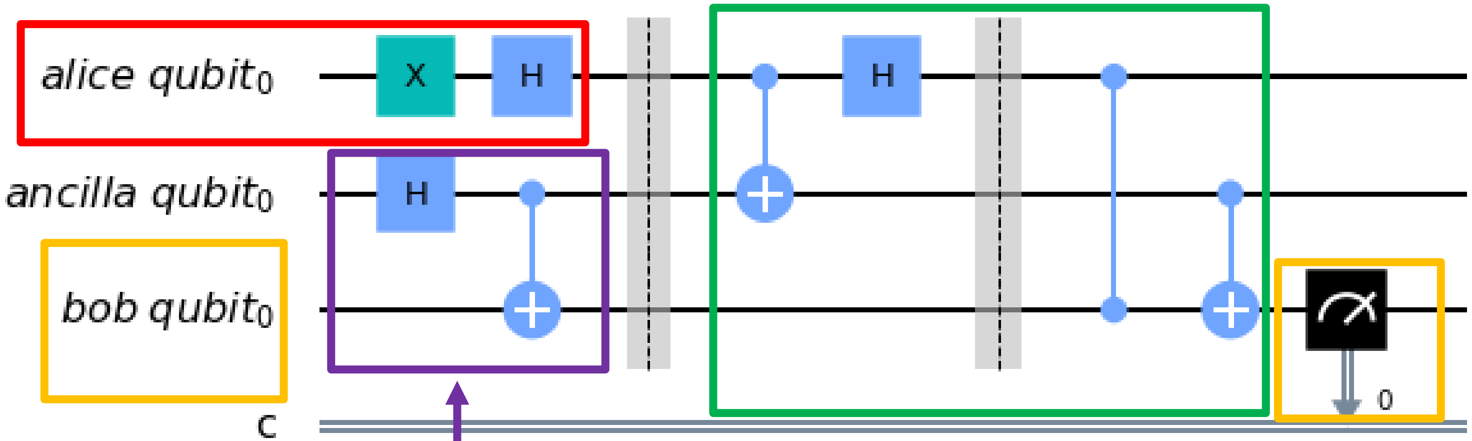
General Idea of the Protocol



IBMQ circuits



5 bit random number generator

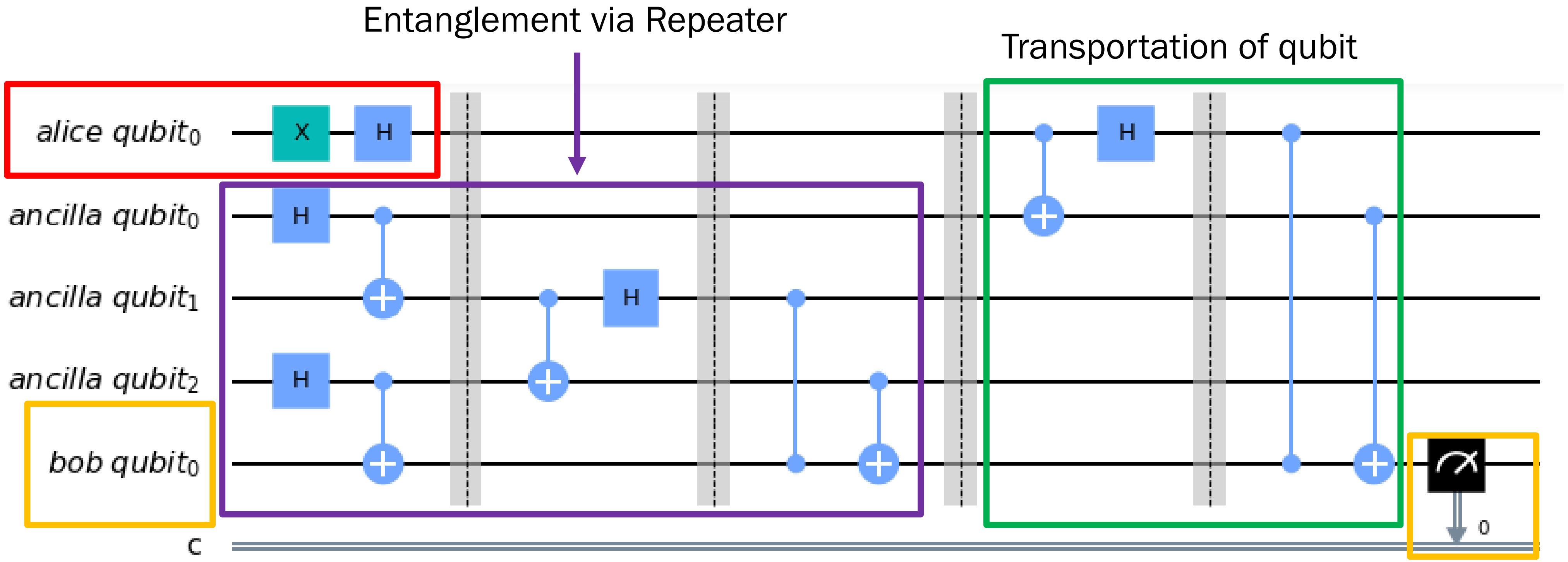


Transportation of qubit

1 qubit transmission

Entanglement

IBMQ circuits

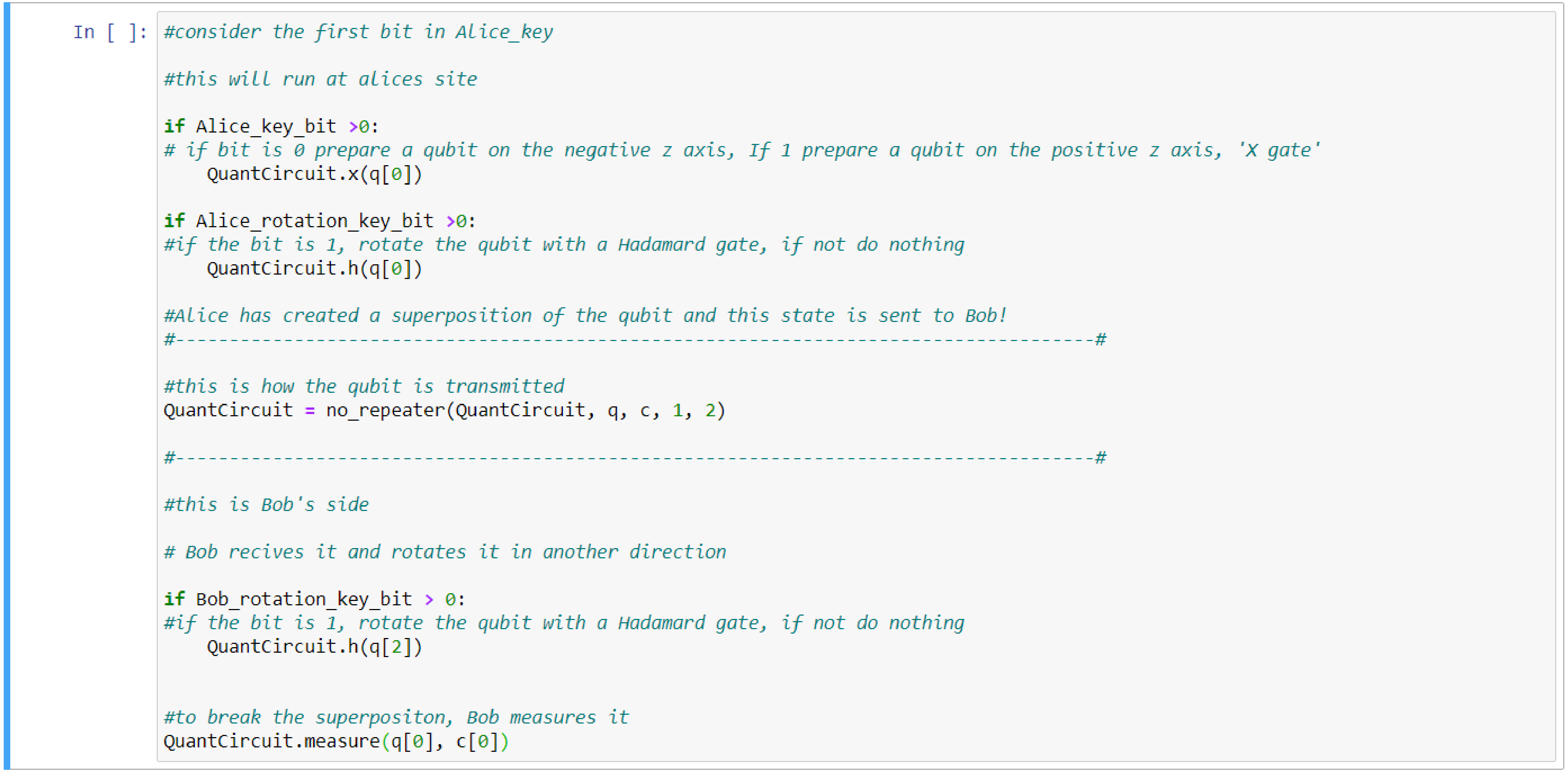


Entanglement via Repeater

Transportation of qubit

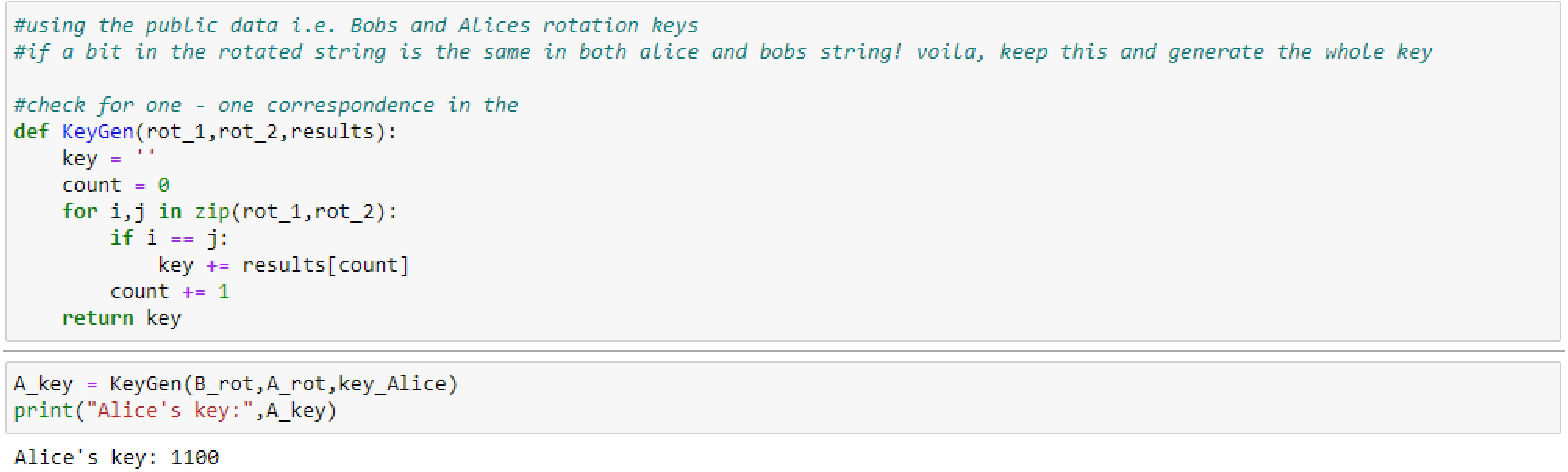
1 qubit transmission

Code to generate superposition



```
#consider the first bit in Alice_key

#this will run at alices site

if Alice_key_bit >0:
# if bit is 0 prepare a qubit on the negative z axis, If 1 prepare a qubit on the positive z axis, 'X gate'
    QuantCircuit.x(q[0])

if Alice_rotation_key_bit >0:
#if the bit is 1, rotate the qubit with a Hadamard gate, if not do nothing
    QuantCircuit.h(q[0])

#Alice has created a superposition of the qubit and this state is sent to Bob!
#---------------------------------------------------------------------------------------#

#this is how the qubit is transmitted
QuantCircuit = no_repeater(QuantCircuit, q, c, 1, 2)

#---------------------------------------------------------------------------------------#

#this is Bob's side

# Bob recives it and rotates it in another direction

if Bob_rotation_key_bit > 0:
#if the bit is 1, rotate the qubit with a Hadamard gate, if not do nothing
    QuantCircuit.h(q[2])


#to break the superpositon, Bob measures it
QuantCircuit.measure(q[0], c[0])
```

Code to generate the key



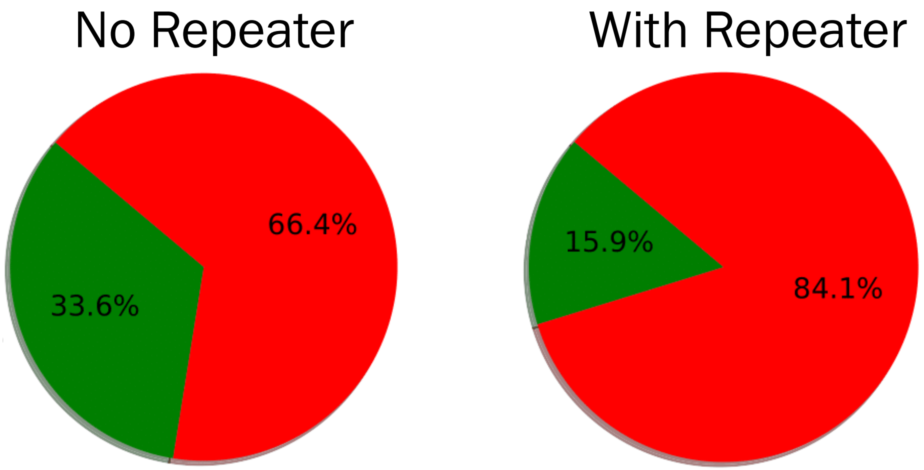
```
#using the public data i.e. Bobs and Alices rotation keys
#if a bit in the rotated string is the same in both alice and bobs string! voila, keep this and generate the whole key

#check for one - one correspondence in the
def KeyGen(rot_1,rot_2,results):
    key = ''
    count = 0
    for i,j in zip(rot_1,rot_2):
        if i == j:
            key += results[count]
        count += 1
    return key
```

```
A_key = KeyGen(B_rot,A_rot,key_Alice)
print("Alice's key:",A_key)
```

```
Alice's key: 1100
```

Simulation Results from IBMQ:
Without Error Mitigation

Number of simulation = 700
Shots = 1

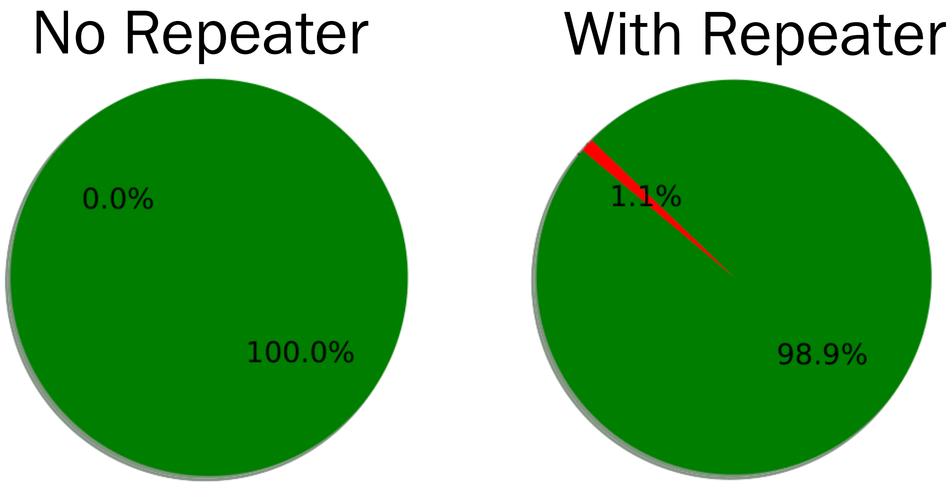


🟩 - Same Keys (successful)
🟥 - Different Keys (unsuccessful)

Simulation Results from IBMQ:

With Error Mitigation

Number of simulation = 700
Shots = 1025

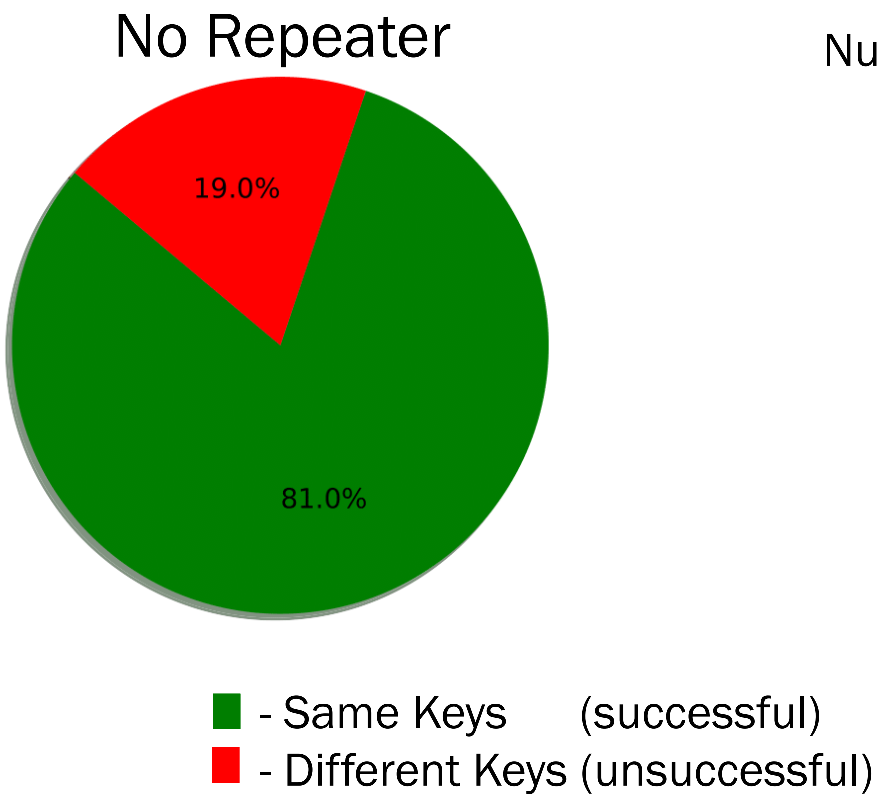


No Repeater

With Repeater

🟩 - Same Keys (successful)
🟥 - Different Keys (unsuccessful)

Simulation Results from IBMQ:
With Error Mitigation



Number of simulation = 147
Shots = 5

Limitations and Challenges

- Sending qubits between different hardware is not possible
  - Communication should happen on one hardware
- Message is sent qubitwise
  - Job execution on real hardware needs to be optimized
- Transmitting long messages takes long queueing time
  - Only a short message is sent

References

www.github.com/Qiskit/qiskit-tutorials

www.qiskit.org/textbook

A Survey of the Prominent Quantum Key Distribution Protocols, Mart Haitjema, Washington University in St. Louis, 2007

Quantum Computation and Quantum Information, Nielsen & Chuang, Cambridge University Press, 2010.

Thank You