**Importing libraries:**

from qiskit import \*

import numpy as np

import matplotlib.pyplot as plt

from qiskit.visualization import plot\_histogram

**Creating Quantum Registers and Circuits:**

n = 7

qr = QuantumRegister(n, name='qr')

cr = ClassicalRegister(n, name='cr')

qc = QuantumCircuit(qr, cr, name='QC')

qr2 = QuantumRegister(n, name='qr')

cr2 = ClassicalRegister(n, name='cr')

qc2 = QuantumCircuit(qr2, cr2, name='QC')

**Quantum Circuit for Key Generation:**

for i in range(n):

qc2.h(i)

qc2.measure(qr2, cr2)

**Executing Quantum Circuit for Key Generation:**

simulator = Aer.get\_backend('qasm\_simulator')

execute(qc2, backend=simulator)

result = execute(qc2, backend=simulator).result()

k = list(result.get\_counts(qc2))[0]

print(k)

**Generating a Random Key for Alice:**

alice\_key = np.random.randint(0, 2\*\*n)

alice\_key = np.binary\_repr(alice\_key, n)

print(alice\_key)

**Alice Encoding:**

for i in range(len(alice\_key)):

if alice\_key[i] == '1':

qc.x(qr[i])

**Basis Selection for Bob:**

B = []

for i in range(len(alice\_key)):

if 0.5 < np.random.random():

qc.h(qr[i])

B.append("H")

else:

B.append("S")

pass

**Measurement and Basis Printing:**

qc.barrier()

print("Alice Basis", B)

**Executing Quantum Circuit for Alice and Bob:**

qc.draw(output='mpl')

simulator = Aer.get\_backend('qasm\_simulator')

execute(qc, backend=simulator)

result = execute(qc, backend=simulator).result()

print("Bob key:", list(result.get\_counts(qc))[0])

**Sifting the Keys:**

def sifted\_key(A\_basis, B\_basis, key):

correct\_basis = []

sifted\_key = ''

for i in range(len(A\_basis)):

if A\_basis[i] == B\_basis[i]:

correct\_basis.append(i)

sifted\_key += key[i]

else:

pass

return sifted\_key, correct\_basis

a = sifted\_key(B, C, alice\_key)

print("sifted key", a[0])

print("Basis", a[1])

BB84\_key = a[0]

**Conversion Functions:**

def wordToBV(s):

# Conversion of text to binary

a\_byte\_array = bytearray(s, "utf8")

byte\_list = []

for byte in a\_byte\_array:

binary\_representation = bin(byte)

byte\_list.append(binary\_representation[9 - n:])

circuit\_array = []

length = len(byte\_list)

for i in range(length):

s = byte\_list[i]

bv\_circuit = QuantumCircuit(n + 1, n)

bv\_circuit.h(n)

bv\_circuit.z(n)

for i in range(n):

bv\_circuit.h(i)

bv\_circuit.barrier()

s = s[::-1]

for q in range(n):

if s[q] == '0':

bv\_circuit.i(q)

else:

bv\_circuit.cx(q, n)

bv\_circuit.barrier()

for i in range(n):

bv\_circuit.h(i)

for i in range(n):

bv\_circuit.measure(i, i)

circuit\_array.append(bv\_circuit)

return circuit\_array

**Building Quantum Circuits for Message Encoding:**

circuit\_to\_run = wordToBV('Qiskit') # Secret Msg

circuit\_to\_run[0].draw(output='mpl')

**Executing Quantum Circuits for Message Encoding:**

backend = BasicAer.get\_backend('qasm\_simulator')

shots = 4096

results = execute(circuit\_to\_run[::-1], backend=backend, shots=shots).result()

answer = results.get\_counts()

plot\_histogram(answer)

**Encrypting the Message:**

def encrypt(BB84\_key, letter):

b = int(BB84\_key, 2)

x = ord(letter)

return format(b ^ x, "b")

secret\_msg = 'qiskit'

L = []

for c in secret\_msg:

L.append(encrypt(BB84\_key, c))

**Steganography Encoding:**

def stega\_encoder(LM, carrier\_msg):

message = ""

size = len(LM[0])

i = 0

for j, bitstring in enumerate(LM):

for k, digit in enumerate(bitstring):

while (not carrier\_msg[i].isalpha()):

message += carrier\_msg[i]

i += 1

if digit == "1":

letter = carrier\_msg[i].upper()

message += letter

else:

message += carrier\_msg[i]

i += 1

if i < len(carrier\_msg):

message += carrier\_msg[i:]

return message

carrier\_msg = 'aaaa aaa aaa aaaa aaaaaaa aaaaaaa aaaaaaa aaabaaa'

new\_carrier\_msg = stega\_encoder(L, carrier\_msg)

new\_carrier\_msg

**Decoding the Secret Message:**

def stega\_decoder(new\_carrier\_msg, BB84\_key):

b = int(BB84\_key, 2)

message = ""

bitstring = ""

for char in new\_carrier\_msg:

if char.isalpha():

if char.isupper():

bitstring += "1"

else:

bitstring += "0"

if len(bitstring) == 7:

x = int(bitstring, 2)

message += chr(b ^ x)

bitstring = ""

return message

stega\_decoder(new\_carrier\_msg, BB84\_key)