Introduction to Quantum Information and Quantum Machine Learning

Laboratory - class 3

Dr Gustaw Szawioła, docent PUT
D. Sc. Eng. Przemysław Głowacki

BB84 protocol simulation

1. BB84 protocol algorithm

BB84 protocol – encoding and decoding states

Charles Bennett
born 1943
Harvard University
IBM Fellow

Gilles Brassard
born April 20, 1955
Université de Montréal

The **BB84** protocol, introduced by **Charles Bennett** and **Gilles Brassard** in 1984, is a **prepare-and-measure** quantum key distribution (QKD) scheme. In this method, one participant (typically called Alice) encodes information by preparing specific quantum states, while the other participant (commonly called Bob) receives and measures those states.

BB84 protocol – encoding and decoding states

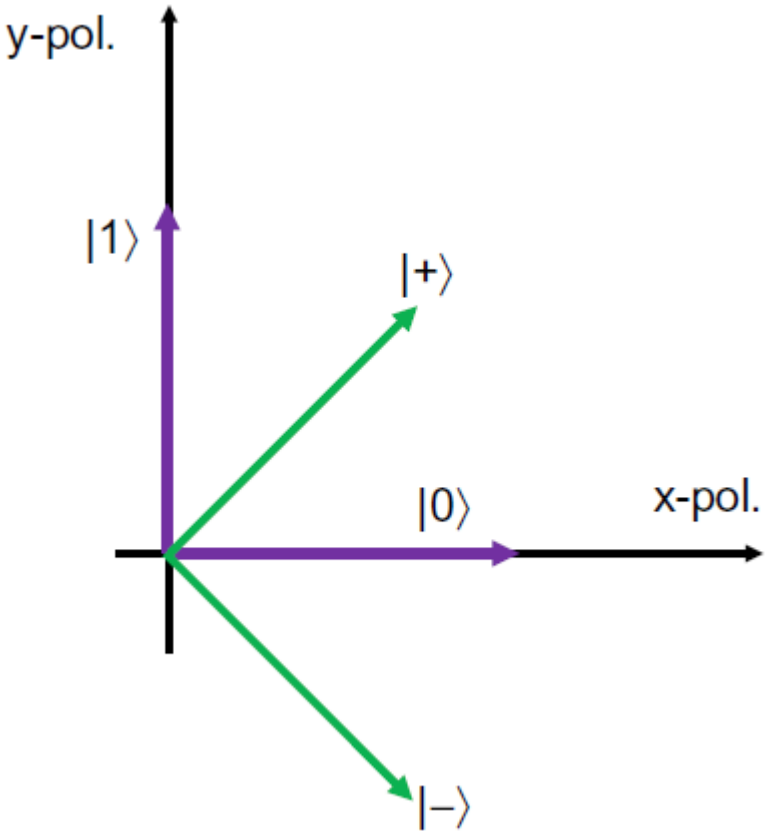
x_a y_a

$$|\psi_{00}\rangle = |0\rangle,$$

$$|\psi_{10}\rangle = |1\rangle,$$

$$|\psi_{01}\rangle = |+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}},$$

$$|\psi_{11}\rangle = |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$



$$|\psi_{00}\rangle = |\uparrow\rangle$$

$$|\psi_{10}\rangle = |\rightarrow\rangle$$

$$|\psi_{01}\rangle = |\nearrow\rangle = \frac{1}{\sqrt{2}}|\uparrow\rangle + \frac{1}{\sqrt{2}}|\rightarrow\rangle$$

$$|\psi_{11}\rangle = |\searrow\rangle = \frac{1}{\sqrt{2}}|\uparrow\rangle - \frac{1}{\sqrt{2}}|\rightarrow\rangle$$

The physical implementation of qubits proposed by the authors of the protocol are single photons with linear polarizations in the standard basis 0^0 , 90^0 and in the skew polarization basis 45^0, 135^0

BB84 protocol

Step	Alice	Transmission	Bob
1	Bit sequence x_A	1 1 0 0 1 0 1 0 1	
	Bit sequence y_A	1 0 0 0 0 1 1 1 1	
2	Preparation basis	✕ + + + + ✕ ✕ ✕ ✕	
	Qubit sequence	↘ ↕ ↔ ↔ ↕ ↗ ↘ ↗ ↘	
4		1 1 0 1 0 0 1 1 0	Bit sequence y_B
		✕ ✕ + ✕ + + ✕ ✕ +	Measurement basis
		1 1 0 1 1 1 1 0 1	Measured bit values x_B
6	Basis matching $(y_A = y_B?)$	✓ ✗ ✓ ✗ ✓ ✗ ✓ ✓ ✗	Basis matching $(y_A = y_B?)$
7	Sifted key x'_A	1 – 0 – 1 – 1 0 –	Sifted key $x'_B = x'_A$

Example of encryption key generation

x_A =q[1]	0	1	0	1	0	0	1	1	0	0
y_A =q[2]	0	0	0	0	1	1	0	0	1	0
y_B =q[3]	1	1	0	1	1	0	0	0	1	1
x_B =q[0]	1	0	0	1	0	0	1	1	0	0
Key match	x	x	0	x	0	x	1	1	0	x

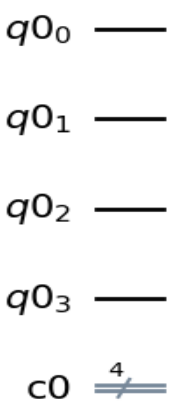
2. Quantum circuit - coding on Alice's side

Importing standard libraries (qiskit 2.2.1, Python 3.13.8)

```
import math
from numpy import pi
from qiskit import *
from qiskit.visualization import *
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from qiskit_aer import Aer
from qiskit.compiler import transpile
```

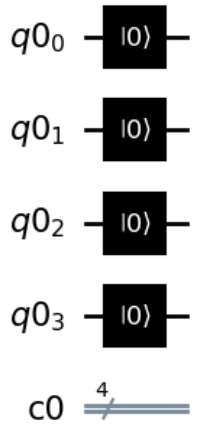
Creating quantum, classical registers and a quantum circuit

```
1. n0=4 # Number of qubits and bits
2. q0 = QuantumRegister(n0) # Quantum register
3. c0 = ClassicalRegister(n0) # Classical register
4. circuit0 = QuantumCircuit(q0, c0) # Quantum
   algorithm - quantum circuit
5. circuit0.draw(output='mpl') # Sketch of a quantum
   circuit
```



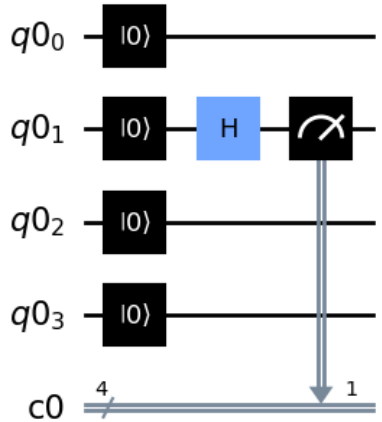
Initializing the initial states of individual quantum registers

```
1.  # Qubit state initialization "0"
    circuit0.reset([q0[0],q0[1],q0[2],q0[3]])
2.  # Sketch of a quantum circuit
    circuit0.draw(output='mpl')
```



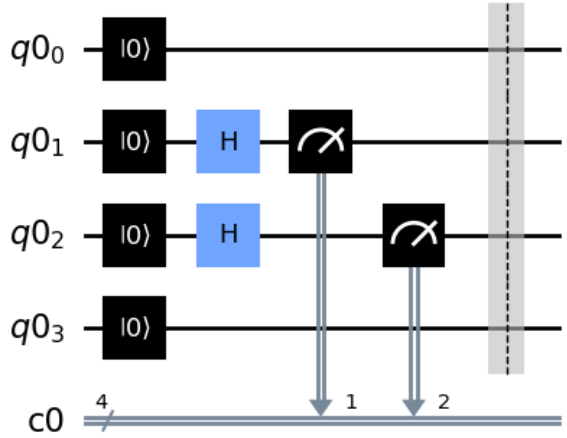
Generation of random number x_A

```
1.  circuit0.h(q0[1])
2.  circuit0.measure(q0[1],c0[1])
    # Sketch of a quantum circuit
1.  circuit0.draw(output='mpl')
```



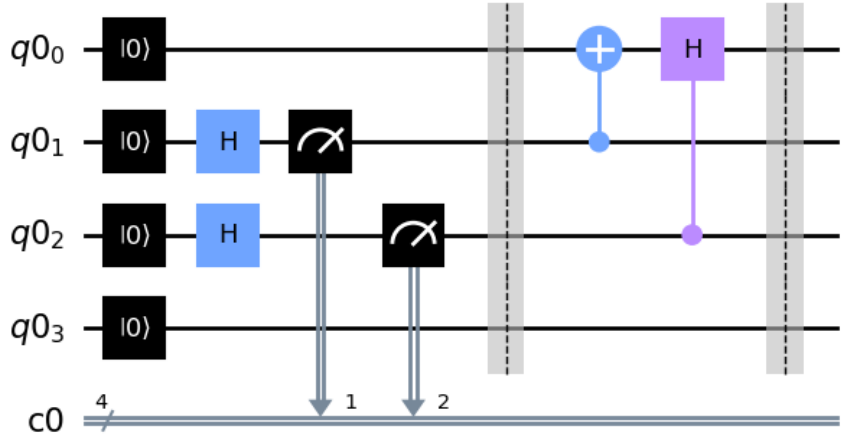
Generation of random number y_A

```
1.  circuit0.h(q0[2])
2.  circuit0.measure(q0[2],c0[2])
3.  circuit0.barrier(q0[0], q0[1], q0[2],
    q0[3])
4.  circuit0.draw(output='mpl')
```



Information coding by Alice

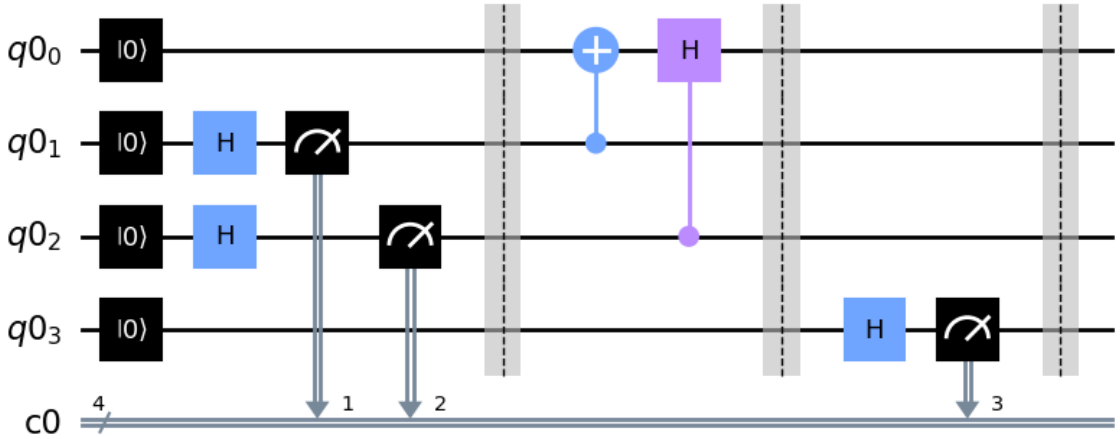
```
1.  circuit0.cx(q0[1],q0[0])
2.  circuit0.ch(q0[2],q0[0])
3.  circuit0.barrier(q0[0], q0[1], q0[2], q0[3])
4.  circuit0.draw(output='mpl')
```



3. Quantum circuit - decoding on Bob's side

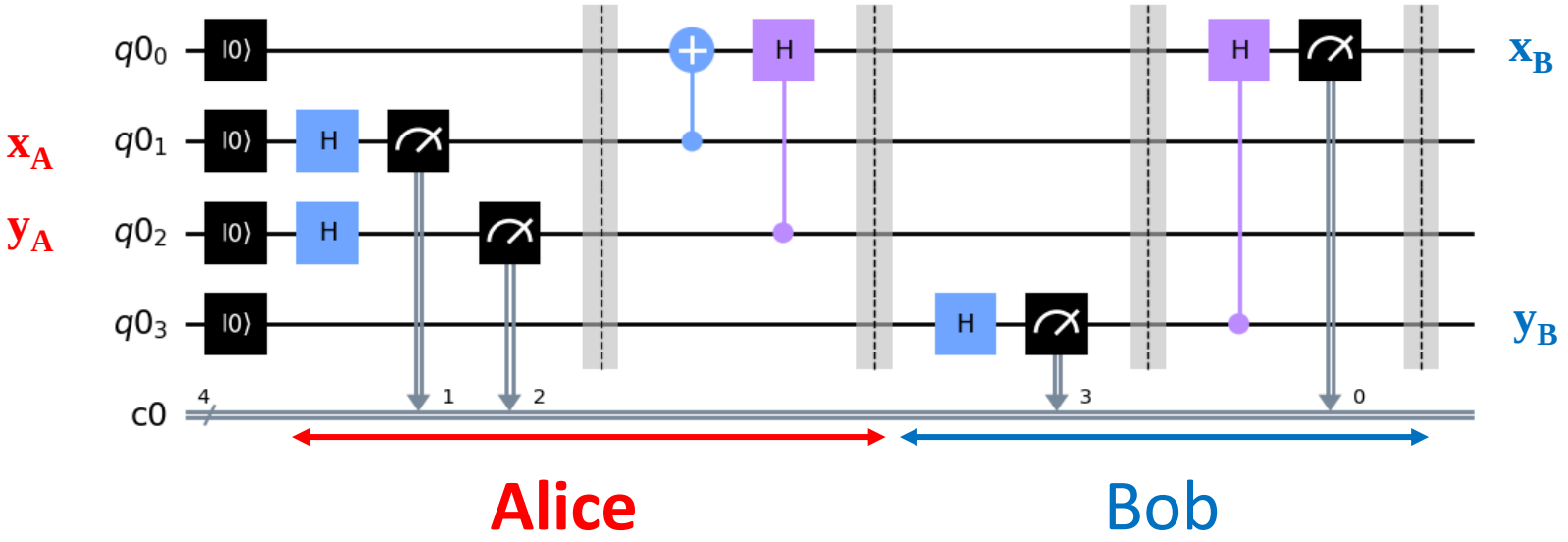
Generation of random number y_B

```
1. circuit0.h(q0[3])
2. circuit0.measure(q0[3],c0[3])
3. circuit0.barrier(q0[0], q0[1], q0[2],
   q0[3])
4. circuit0.draw(output= 'mpl')
```



Bob decoding information

```
1. circuit0.ch(q0[3],q0[0])
2. circuit0.measure(q0[0], c0[0]) # Checking qubit
   states - quantum measurement on qubit "0„
3. circuit0.barrier(q0[0], q0[1], q0[2], q0[3])
4. circuit0.draw(output= 'mpl')
   # Sketch of a quantum circuit
```



4. Quantum circuit startup and simulation

Selecting a simulator and test run

```
# Selecting a quantum simulator
backend = Aer.get_backend('qasm_simulator')
# Performing quantum calculations
# Transpiling the circuit to a backend compatible form
compiled_circuitX = transpile(circuit0, backend)
# Running the circuit on the simulator
job_sim0 = backend.run(compiled_circuitX, shots=1)
sim_result0 = job_sim0.result()
wynik=sim_result0.get_counts(circuit0)
# Numerical presentation of measurement results
print(wynik)
```

{'0100': 1} $\xleftarrow{format}$ {'$y_B\ y_A\ x_A\ x_B$' : 1}

Generation of a sequence of numbers x_B (including the remaining numbers)

```
sample=10
1. bit=[]
2. for kk in range(sample):
3.     compiled_circuitX = transpile(circuit0,
        backend)
4.     job_sim0 = backend.run(compiled_circuitX,
        shots=1)
5.     sim_result0 = job_sim0.result()
6.     wynik=sim_result0.get_counts(circuit0)
7.     xA=int(list(wynik.keys())[0][2])
8.     yA=int(list(wynik.keys())[0][1])
9.     yB=int(list(wynik.keys())[0][0])
10.    xB=int(list(wynik.keys())[0][3])
11.    print(wynik,"->",[xA,yA,yB,xB])
12.    bit.append([xA,yA,yB,xB])
```

Printed results

```
{'0000': 1} -> [0, 0, 0, 0]
{'0011': 1} -> [1, 0, 0, 1]
{'0000': 1} -> [0, 0, 0, 0]
{'0000': 1} -> [0, 0, 0, 0]
{'1100': 1} -> [0, 1, 1, 0]
{'1111': 1} -> [1, 1, 1, 1]
{'0111': 1} -> [1, 1, 0, 1]
{'0101': 1} -> [0, 1, 0, 1]
{'1100': 1} -> [0, 1, 1, 0]
{'0011': 1} -> [1, 0, 0, 1]
12.print(bit)
[[0, 0, 0, 0], [1, 0, 0, 1], [0, 0, 0, 0],
[0, 0, 0, 0], [0, 1, 1, 0], [1, 1, 1, 1],
[1, 1, 0, 1], [0, 1, 0, 1], [0, 1, 1, 0],
[1, 0, 0, 1]]
```

Key sifting

```
1. # Key sifting
2. kluczA=[]
3. kluczB=[]
4. for bb in bit:
5.     if bb[1]==bb[2]:
6.         kluczA.append(bb[0])
7.         kluczB.append(bb[3])
8.         print('yA=',bb[1], 'yB=',bb[2],
       '->','xA=',bb[0],',xB=',bb[3])
9. print('kluczA=',kluczA)
10.print('kluczB=',kluczB)
```

Printed results

```
• yA= 0 ,yB= 0 -> xA= 0 ,xB= 0
• yA= 0 ,yB= 0 -> xA= 1 ,xB= 1
• yA= 0 ,yB= 0 -> xA= 0 ,xB= 0
• yA= 0 ,yB= 0 -> xA= 0 ,xB= 0
• yA= 1 ,yB= 1 -> xA= 0 ,xB= 0
• yA= 1 ,yB= 1 -> xA= 1 ,xB= 1
• yA= 1 ,yB= 1 -> xA= 0 ,xB= 0
• yA= 0 ,yB= 0 -> xA= 1 ,xB= 1
• kluczA= [0, 1, 0, 0, 0, 1, 0, 1]
• kluczB= [0, 1, 0, 0, 0, 1, 0, 1]

• kluczA==kluczB
True
```

Key sifting

The final result of your report should be a table with the length of the encryption key (secret key) depending on the use of n bits for encoding

Sample = n	Number of bits in Sifted key (n bits in agreed secret key)		
	Test 1	Test 2	Test 3
16	7	8	5
32	18	17	19
64	30	34	28
128	60	64	67
256	131	117	124

The End