BB84 プロトコル

@0917laplace

はじめに

本スライドは、BB84 プロトコルについて解説するものである。

以下の Qiskit Textbook を主に参照する。

`https://qiskit.org/textbook/ja/ch-algorithms/quantum-key-distribution.html`

非クローン定理

Theorem (非クローン定理)

$\mathscr{H}$ を有限次元 *Hilbert* 空間で，$\mathscr{S}_1, \mathscr{S}_2$ を $\mathscr{H}$ に付随する量子系とする．ρ_2 を $\mathscr{S}_2$ の量子状態で固定して，$\mathscr{S}_1$ の任意の量子状態 ρ_1 に対して，$U(\rho_1 \otimes \rho_2)U^\dagger = \rho_1 \otimes \rho_1$ なる $\mathscr{H} \otimes \mathscr{H}$ 上の *unitary* 演算子 U は存在しない．

状況設定

$\{0,1\}$ に対して,

$$|0\rangle_{\oplus} = |0\rangle,\ |1\rangle_{\oplus} = |1\rangle,\ |0\rangle_{\otimes} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle),\ |1\rangle_{\otimes} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

とする. BB84 プロトコルでは, Alice が偏光フィルターを通した光子 1 つに $\{0,1\}$ のビット情報をランダムに載せて, Bob に送信することを考える.
偏光フィルターは $\oplus, \otimes$ のいずれかであり, 例えば, 0 の bit 情報に $\otimes$ のフィルターを通した後の光子の量子状態は $|0\rangle_{\otimes}$ とみなす.
Bob は, ランダムに $\oplus, \otimes$ からなる検出器を使って, Alice のメッセージを測定する.

以下では, 偏光フィルター・検出器を共に基底とみなす.
また, 実際には, 光子を光波とみて, 波の振動の向きで 4 つの量子状態を表現できる.

QKD の暗号化の手順

2) と 3) の間で Eve が盗聴しようとすると、その測定で量子ビットの状態が変化するため、盗聴を検知できる。

1) Alice はランダムなビット列とランダムな基底を選ぶ
2) Alice は 1) のビットと基底を使って、量子ビット列上にエンコードし、Bob へ送信する
3) Bob はランダムな基底を使って、Alice のメッセージを測定する
4) Alice と Bob は基底を公開し、秘密鍵を共有する
5) それぞれのビットをランダムサンプリングして比較し、プロトコルが正常か確認する

3) の補足

例えば，Alice の $|0\rangle_{\otimes}$ に対して，Bob が $\oplus$ の基底を選んだ場合，$|0\rangle$ を測定する確率は $\frac{1}{2}$ であり，$|1\rangle$ を測定する確率も $\frac{1}{2}$ である．つまり，Alice と Bob の選んだ基底が異なる場合でも，確率 $\frac{1}{2}$ でビット情報は一致することを表している．

Eve が盗聴する場合

Eve もランダムに基底を選んで，盗聴する場合を考える．例えば，Alice が $|0\rangle_{\oplus}$ を送信し，Bob が基底として $\oplus$ を選んでいたとする．このとき，Eve は $\otimes$ を基底として観測した場合，量子ビット $|0\rangle$ を確率 $\frac{1}{2}$ で測定し，$|1\rangle$ を確率 $\frac{1}{2}$ で測定する．Eve が $|1\rangle$ を測定した場合，Eve は $|1\rangle_{\otimes}$ を Bob へ送信することになり，Bob は $\oplus$ で測定するため，量子ビット $|0\rangle$ を確率 $\frac{1}{2}$ で測定し，$|1\rangle$ を確率 $\frac{1}{2}$ で測定する．

盗聴がなければ，Bob は確率 1 で $|0\rangle$ を測定するため，Alice と Bob の基底が一致しても，観測ビットが確率 $\frac{1}{2}$ で異なるため，盗聴を検知することができる．

Eve は測定した量子状態をそのまま Bob に送るほかない．Alice と Eve が選んだ基底が直交しない（未知の量子状態）場合に，Alice から受け取った量子状態をそのまま Bob へ送ることは，非クローン定理からできないためである．

前処理

- 必要なパッケージのインポート
- 100 量子ビット長のメッセージを送信する

```
## 前処理
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

## 100 量子ビットで固定
n = 100
## seed は 0 で固定
np.random.seed(seed=0)
```

Step1(Alice はランダムなビット列と基底を選ぶ)

```
## Step 1
### Alice はランダムなビット列とランダムな基底を選ぶ

alice_bits = randint(2, size=n)
alice_bases = randint(2, size=n)
```

Step2(Alice はメッセージをエンコードする)

```
## Step 2
### Alice は量子ビット列上にエンコードし、Bob に送信する

def encode_message(bits, bases):
    message = []
    for i in range(n):
        qc = QuantumCircuit(1,1)
        if bases[i] == 0: # Prepare qubit in Z-basis
            if bits[i] == 0:
                pass
            else:
                qc.x(0)
        else: # Prepare 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

message = encode_message(alice_bits, alice_bases)
```

Step3(Bob は Alice からのメッセージを測定する)

```
## Step 3
### Bob はランダムな基底を使って、Alice のメッセージを測定する

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


bob_bases = randint(2, size=n)
bob_results = measure_message(message, bob_bases)
```

Step4(Alice と Bob は鍵蒸留処理を行う)

```
## Step 4
### Alice と Bob は基底を公開し、秘密鍵を共有する

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


bob_key = remove_garbage(alice_bases, bob_bases, bob_results)
alice_key = remove_garbage(alice_bases, bob_bases, alice_bits)
```

Step5(Alice と Bob は鍵を比較する)

```
## Step 5
### Alice と Bob は鍵のランダムサンプリングを比較する

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


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))
```

盗聴ありの場合

Step2 と Step3 の間で、Eve が Alice からのメッセージを以下のように盗聴したとする。

```
## 盗聴
### Eve はランダムな基底を使って、Alice のメッセージを盗聴（測定）
    する

eve_bases = randint(2, size=n)
intercepted_message = measure_message(message, eve_bases)
```

このとき、Step5 でのサンプリング後の比較は次のようになり、盗聴を検知できている。

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