Introduzione alla meccanica quantistica per il quantum computing

Quantum Key Distribution - Protocollo BB84

Università di Verona

Imbriani Paolo - VR500437
Irimie Fabio - VR501504

Profssa. Daffara Claudia

29 aprile 2025

Indice

1 Introduzione **3**

2 Cos'è BB84? **3**
2.1 Requisiti 3
2.2 Funzionamento 4

3 Scenario concreto **4**
3.1 Codifica 4
3.2 Trasmissione 7

1 Introduzione

Fabio e Paolo vogliono instaurare un canale di comunicazione sicuro, in modo da poter inviare messaggi privati senza che un eventuale attaccante possa intercettarli. I due si affidano alla crittografia, che permette di codificare i messaggi in modo che solo il destinatario possa decifrarli.

I due potrebbero utilizzare un metodo classico, come ad esempio il cifrario di Cesare che prevede di sostituire ogni lettera con una lettera che si trova un certo numero di posizioni più avanti nell'alfabeto. Ad esempio:

$$\begin{aligned} \mathrm{A} &\to \mathrm{D} \\ \mathrm{B} &\to \mathrm{E} \\ \mathrm{C} &\to \mathrm{F} \\ &\vdots \\ \mathrm{Z} &\to \mathrm{C} \end{aligned}$$

Immaginiamo che Fabio voglia inviare un messaggio a Paolo, per far sì che Paolo possa decifrare il messaggio, Fabio deve comunicargli la chiave, ovvero il numero di posizioni da spostare nell'alfabeto. Questo metodo è facilmente intercettabile, infatti l'attaccante può semplicemente provare tutte le possibili chiavi fino a trovare quella giusta. Questo tipo di crittografia è detta simmetrica, in quanto sia il mittente che il destinatario devono possedere la chiave per cifrare e decifrare il messaggio.

2 Cos'è BB84?

Per far sì che Fabio e Paolo possano creare un canale di comunicazione sicuro, devono trovare un modo per scambiarsi la chiave in segreto. Però anche in questo caso, la condivisione della chiave deve essere fatta in un canale sicuro, che è il problema che vogliamo risolvere dal principio!

Il protocollo BB84 è un protocollo di *Quantum Key Distribution* che risolve questo problema. Se Fabio usasse questo protocollo per trasmettere la chiave a Paolo, loro potrebbero sapere con *quasi* assoluta certezza se la chiave è stata intercettata o meno.

- Se la chiave **non** è stata intercettata, Fabio e Paolo avrebbero una chiave segreta per instaurare un canale di comunicazione sicuro.
- Se la chiave **è** stata intercettata, Fabio e Paolo possono decidere di non utilizzare la chiave e di ripetere il protocollo.

2.1 Requisiti

Per funzionare, il protocollo BB84 ha bisogno dei seguenti requisiti:

- Sia Fabio che Paolo devono avere accesso al proprio computer quantistico.
- Devono avere un canale di trasmissione capace di trasmettere qubit. Questo potrebbe essere qualche tipo di cavo a fibra ottica capace di trasmettere fotoni polarizzati.

- Devono avere un canale di comunicazione classico. Siccome è impossibile assicurare una sicurezza perfetta, bisogna assumere che qualsiasi canale classico può essere intercettato.

2.2 Funzionamento

I passaggi del protocollo sono i seguenti:

1. Fabio crea una stringa casuale di bit, e per ogni bit, lui sceglie casualmente una base in cui codificarlo.
2. Fabio codifica i bit in qubit usando la base scelta, e invia i qubit al computer quantistico di Paolo attraverso un canale di comunicazione quantistico.
3. Anche Paolo sceglie casualmente una base secondo cui decodificare ogni qubit ricevuto. Quindi misura ogni qubit nella base che ha scelto.
4. Fabio utilizza un canale di comunicazione classico per dire a Paolo che basi ha scelto per la codifica. Poi comunica anche i primi bit della chiave non codificati.
5. Paolo analizza questi primi bit per determinare se un intercettatore (Amos) è riuscito ad entrare nel canale di comunicazione quantistico e intercettare i qubit che Fabio gli ha inviato.
6. Si distinguono due casi:
   - Se Amos **non** ha intercettato i qubit, Fabio e Paolo possono considerare tutti i qubit che hanno scelto **in comune**, cioè con la stessa base, come la loro chiave segreta.
   - Se Amos **ha** intercettato i qubit, Fabio e Paolo devono ripetere il processo da capo.

3 Scenario concreto

Prendiamo in considerazione un esempio concreto per capire meglio il protocollo BB84, in cui Fabio è il mittente e Paolo è il destinatario. Questo esempio è scritto in Python e utilizza la libreria `qiskit` per simulare il computer quantistico. Le librerie da importare sono le seguenti:

```
1 from qiskit import *
2 import random
```

3.1 Codifica

Per prima cosa Fabio deve scegliere una sequenza di bit e basi da utilizzare per codificare i bit in qubit. Un qubit può essere rappresentato come un vettore sulla sfera di Bloch, che è una rappresentazione geometrica di uno stato quantistico. Ogni asse può essere considerato come una possibile base. Quindi se un vettore punta in alto, il qubit è codificato in $|0\rangle$, se punta in basso è codificato in $|1\rangle$. L'asse verticale è chiamato *asse Z*. Nella base Z possiamo codificare lo 0 come $|0\rangle$ (figura 1 a sinistra) e l'1 come $|1\rangle$ (figura 1 a destra).

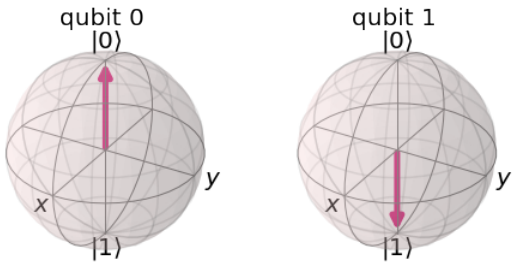

Figura 1: Base Z

Un altro modo di codificare i qubit è utilizzare la base X, in cui lo 0 è rappresentato da $|+\rangle$ (figura 2 a sinistra) e l'1 da $|-\rangle$ (figura 2 a destra).

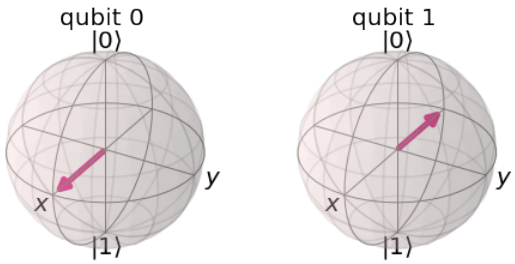

Figura 2: Base X

Il primo passo per Fabio è generare casualmente una stringa di bit di lunghezza arbitraria, ad esempio 300 bit.

```
KEY_LENGTH = 300 # Lunghezza della chiave
random.seed(0)

# Genera una chiave casuale di lunghezza KEY_LENGTH
fabio_bits = ""
for i in range(KEY_LENGTH):
    randBit = random.randint(0, 1) # Numero casuale 0 o 1
    fabio_bits += str(randBit) # Aggiungi il bit alla chiave

print("I bit che Fabio invia sono: " + fabio_bits[:10] + "...")
```

```
I bit che Fabio invia sono: 1011101010...
```

Successivamente Fabio sceglie una base per ogni bit (o in base Z o in base X).

```
def generate_random_bases(num_of_bases):
    """Questa funzione seleziona una base casuale per ogni bit"""
    bases_string = ""
    for i in range(num_of_bases):
        randBasis = random.randint(0, 1)

        if randBasis == 0:
            bases_string += "Z"
        else:
            bases_string += "X"

```

```
    return bases_string

# Fabio sceglie casualmente una base per ogni bit
fabio_bases = generate_random_bases(KEY_LENGTH)
print("Le basi che Fabio usa sono: " + fabio_bases[:10] + "...")
```

```
Le basi che Fabio usa sono: XXZZZXZXZZ...
```

Ora Fabio, sul suo computer quantistico, codifica i bit in qubit utilizzando le basi che ha scelto, creando così una stringa di 300 qubit. Successivamente, Fabio invia i qubit al computer quantistico di Paolo attraverso un canale di comunicazione quantistico.

Di default, Qiskit utilizza la base Z per codificare i qubit e tutti i qubit sono inizializzati in $|0\rangle$. Per trasformare il qubit in $|1\rangle$ bisogna applicare una porta X.

Mentre invece, per codificare i qubit in base X si inizia con i corrispondenti $|0\rangle$ e $|1\rangle$ e si applica una porta H (Hadamard) per trasformarli rispettivamente in $|+\rangle$ e $|-\rangle$.

La tabella delle conversioni è la seguente:

Bit	**Base**	**Stato**	**Porta**
0	Z	$\|0\rangle$	-
1	Z	$\|1\rangle$	X
0	X	$\|+\rangle$	H
1	X	$\|-\rangle$	H, X

Usando come riferimento questa tabella, il circuito quantistico per codificare i bit in qubit è il seguente:

```
def encode(bits, bases):
    """La funzione codifica ogni bit nella base data"""

    encoded_qubits = []

    for bit, basis in zip(bits, bases):
        # Crea un circuito quantistico per ogni qubit
        qc = QuantumCircuit(1, 1)

        # Casi possibili
        if bit=="0" and basis == "Z":
            encoded_qubits.append(qc) # Non serve applicare nessuna
     porta

        elif bit=="1" and basis == "Z":
            qc.x(0) # Applica porta X
            encoded_qubits.append(qc)

        elif bit=="0" and basis == "X":
            qc.h(0) # Applica porta H
            encoded_qubits.append(qc)

        elif bit=="1" and basis == "X":
            qc.x(0) # Applica porta X
            qc.h(0) # Applica porta H
            encoded_qubits.append(qc)

    return (encoded_qubits)
```

```
28
29
30 # Codifica i bit di Fabio
31 encoded_qubits = encode(fabio_bits, alice_bases)
32
33 # Print circuits for first 5 qubits.
34 for i in range(5):
35     print(encoded_qubits[i])
36 print("etc.")
```

3.2 Trasmissione