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

1 maggio 2025

Indice

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).

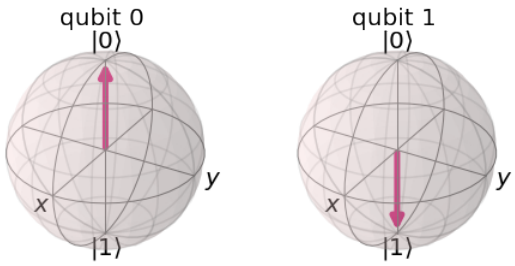

Figura 1: Base Z

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).

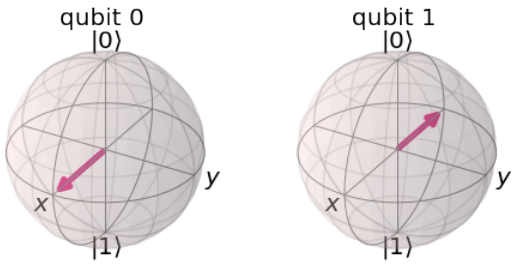

Figura 2: Base X

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

Normalmente, i qubit dovrebbero essere inviati attraverso un canale di comunicazione quantistico, ma essendo che non è possibile avere accesso ad una cosa del genere nel mondo reale, per semplicità Fabio inserirà i qubit codificati in una lista chiamata `CANALE_QUANTISTICO`. Ovviamente, proprio come un cavo a fibra ottica, il canale quantistico può essere intercettato da un attaccante.

```
1 CANALE_QUANTISTICO = encoded_qubits
```

3.3 Misurazione

Il passo successivo per Paolo è quello di ricevere i qubit codificati e misurarli. Prima di tutto, deve scegliere una serie di basi casuali, proprio come ha fatto Fabio in precedenza.

```
1 paolo_bases = generate_random_bases(KEY_LENGTH)
2
3 print("Le basi che Paolo usa sono: " + paolo_bases[:10] + "...")
```

```
1 Le basi che Paolo usa sono: ZXXZZZXZXZ...
```

Successivamente, Paolo deve misurare ogni qubit nella base che ha scelto. In Qiskit, la misurazione di un qubit è rappresentata da una porta M che andremo ad inserire per ogni qubit. Tuttavia, dobbiamo stare attenti: Qiskit non permette di misurare un qubit in una base diversa da quella Z, quindi dobbiamo prima applicare la porta H per poter misurare la base X.

```
1  def measure(qubits, bases):
2  """Questa funzione misura ogni qubit nella base corrispondente
       scelta per esso."""
3
4  bits = ""
5
6  for qubit, basis in zip(qubits, bases):
7
8      #  Misurazione dipendente dalla base
9      if basis == "Z":
10         qubit.measure(0, 0)
11     elif basis == "X":
12         qubit.h(0)
13         qubit.measure(0, 0)
14
15     # Eseguire sul simulatore
16     simulator = Aer.get_backend('qasm_simulator')
17     result = execute(qubit, backend=simulator, shots=1).result()
18     counts = result.get_counts()
19     measured_bit = max(counts, key=counts.get)
```

```
20
21      bits += measured_bit
22
23  return bits
```

```
1  qubits_received = CANALE_QUANTISTICO # Paolo riceve i bit dal
       canale quantistico
2  paolo_bits = measure(qubits_received, paolo_bases)
3
4  print("I primi bit che Paolo ha ricevuto sono: " + bob_bits[:10] +
       "...")
```

```
1  I primi bit che Paolo ha ricevuto sono: 1011101010...
```

3.4 Comparazione

Ora, Fabio deve comunicare a Paolo le basi che ha scelto per codificare i suoi qubit. Lo può fare attraverso qualsiasi canale di comunicazione classico. Il bello di questo protocollo è che non importa se Amos sa quali basi sono state usate. Fabio potrebbe persino pubblicare queste basi su un forum pubblico!


```
1  CANALE_CLASSICO = fabio_bases # Fabio comunica a Paolo le basi che
       ha utilizzato
```

Per ogni qubit dove Fabio e Paolo hanno scelto basi diverse, c'è un 50% di probabilità che la misurazione di Paolo ritorni il qubit errato. Per esempio, se Fabio invia a Paolo in qubit nello stato $|+\rangle$ (quindi un bit 0 codificato nella base X), e Paolo lo misura nella base Z, c'è una probabilità nel 50% di ottenere $|0\rangle$ e il 50% di ottenere $|1\rangle$. Di conseguenza, ogni istanza dove le loro basi non corrispondono è inutile: Paolo deve trovare le basi che ha in comune con Fabio.

```
1  common_bases = [i for i in range (KEY_LENGTH) if CANALE_CLASSICO[i]
       == paolo_bases[i]]
2
3  print("Gli indici delle prime dieci basi in comune sono: " + str(
       common_bases[:10]))
```

```
1  Gli indici delle prime dieci basi in comune sono: [1, 3, 4, 9, 11,
       12, 20, 24, 26, 27].
```

Ora che Paolo sa le basi in comune, può scartare tutti i bit che non corrispondono con le sue basi e mantenere solo quelli che invece condividono.

```
paolo_bits = [paolo_bits[index] for index in common_bases]
```

Inoltre comunica a Fabio quali basi avevano in comune così che anche lui a sua volta, possa scartare il resto dei bit, mantenendo solo quelli che ha in comune con Paolo.

```
CANALE_CLASSICO = common_bases # Paolo comunica a Fabio le basi in
    comune
```

```
fabio_bits = [fabio_bits[index] for index in common_bases]
```

Essendo che entrambi mantengono solo i bit misurati con le basi che condividono *dovrebbero* avere gli stessi bit. Per essere sicuri che questo sia il caso, Fabio annuncierà i primi bit che ha e Paolo dovrebbe avere gli stessi. Ovviamente se Amos provasse ad origliare, anche lui verrebbe a conoscenza dei primi bit, quindi Fabio e Paolo dovranno successivamente scartarli (solo dopo averli confrontati, per essere sicuri che abbiano gli stessi bit come previsto).

```
CANALE_CLASSICO = fabio_bits[:100]

# Paolo controlla se i primi 100 bit di Fabio corrispondono con i
    suoi primi 100 bit
if CANALE_CLASSICO == paolo_bits[:100]:
    print("Paolo e Fabio sembrano avere gli stessi bit.")
else:
    print("Almeno un bit e' risultato diverso.")
```

```
Paolo e Fabio sembrano avere gli stessi bit.
```

Essendo che i primi 100 bit sono uguali, Fabio e Paolo possono essere abbastanza sicuri che anche i rimanenti bit corrispondono. Ora, devono scartare i primi 100 bit, perchè Amos potrebbe averli intercettati.

```
fabio_bits = fabio_bits[100:]
paolo_bits = paolo_bits[100:]
```

A questo punto, i bit rimanenti saranno la **chiave** che verrà usata per creare il canale di comunicazione criptato. Ora possono comunicare in libertà senza avere paura che la loro conversazione venga letta da qualcun altro.

```
key = ""
for bit in fabio_bits: # O paolo_bits essendo che dovrebbero essere
     gli stessi.
    key += bit

print("La chiave segreta e':")
print(str(key))

print("\nLa chiave e' lunga" + str(len(key)) + " bit.")
```

```
La chiave segreta e':
101110010101101010100110010110101010111110101000101010111010101000101000111010
La chiave e' lunga 90 bit.
```

3.5 Intercettazione

Fino ad ora abbiamo considerato che Amos non sia riuscito ad intercettare i qubit, ma cosa succederebbe se invece ci riuscisse? Vediamo il caso nel dettaglio: Come sempre, Fabio genera una sequenza casuale di basi per codificare i suoi bit. Successivamente invia questi qubit a Paolo attraverso il canale quantistico.

```
fabio_bits = ""
for i in range(KEY_LENGTH):
    randBit = random.randint(0, 1)
    fabio_bits += str(randBit)

fabio_bases = generate_random_bases(KEY_LENGTH)

encoded_qubits = encode(fabio_bits, fabio_bases)

CANALE_QUANTISTICO = encoded_qubits
```

3.5.1 Amos ha intercettato i qubit

Questa volta i qubit vengono intercettati da Amos. Vediamo cosa farebbe. Prima di tutto, lui sceglie casualmente una serie di basi per misurare i qubit (dato che non ha idea di quali basi abbia usato Fabio). Poi, esegue le misurazioni. Questo è simile a quello che farebbe Paolo normalmente.

```
qubits_intercepted = CANALE_QUANTISTICO # Intercetta i qubit
amos_bases = generate_random_bases(KEY_LENGTH)
amos_bits = measure(qubits_intercepted, eve_bases)
```

A causa del teorema del no-cloning quantistico, Amos non può semplicemente duplicare esattamente i qubit che stanno venendo inviati sul canale quantistico. Di conseguenza, Paolo non riceverà mai i qubit originali rendendo ovvio che Amos ha intercettato i qubit.

3.5.2 Teorema del no-cloning

Partiamo dalle basi per poter capire al meglio il teorema del no-cloning.
Uno stato quantistico $|\psi\rangle$ può essere rappresentato come una combinazione lineare di stati:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

dove α e β sono numeri complessi tali che $|\alpha|^2 + |\beta|^2 = 1$ (normalizzati). In questo momento si dice che il qubit è in una superposizione di due stati.
Quello che dice il teorema del no-cloning è che non esiste un operatore (o trasformazione) universale unitario U tale che

$$|\psi\rangle \otimes |e\rangle \to |\psi\rangle \otimes |\psi\rangle$$

per qualsiasi input arbitrario (nel nostro caso, sono i bit sono all'interno del canale quantistico) $|\psi\rangle$, dove $|e\rangle$ è uno stato iniziale del sistema da clonare qualsiasi (per esempio $|0\rangle$).
In altre parole **non esiste un processo tale che possa duplicare uno stato quantistico sconosciuto**.

Dimostrazione del teorema del no-cloning

Supponiamo per assurdo che invece *esista* un operatore U tale che

$$U(|\psi\rangle \otimes |e\rangle) = |\psi\rangle \otimes |\psi\rangle$$
$$U(|\phi\rangle \otimes |e\rangle) = |\phi\rangle \otimes |\phi\rangle$$

Ora prendiamo il prodotto interno di entrambi gli output:

$$\langle\psi|\phi\rangle^2 = \langle\psi|\phi\rangle\langle\psi|\phi\rangle$$

Poiché U è un operatore unitario, abbiamo che dovrebbe preservare i prodotti interni. (Le probabilità totale deve rimanere 1 e le operazioni unitarie devono essere reversibili) di conseguenza il prodotto interno tra gli stati di input deve essere uguale al prodotto interno tra gli stati di output:

$$\langle\psi|\phi\rangle = \langle\psi|\phi\rangle^2$$

ricordiamo che:

$$\langle\psi|\phi\rangle = \begin{pmatrix}\psi_1 & \psi_2 & \dots & \psi_n\end{pmatrix} \begin{pmatrix}\phi_1 \\ \phi_2 \\ \vdots \\ \phi_n\end{pmatrix} = \psi_1\phi_1 + \psi_2\phi_2 + \cdots + \psi_n\phi_n$$

L'equazione è vera soltanto se

- $\langle\psi|\phi\rangle = 0$ (stati ortogonali) oppure
- $\langle\psi|\phi\rangle = 1$ (stati identici)

Questo vuol dire che solo stati ortogonali e stati identici possono essere clonati, ma non stati arbitrari. Essendo che la maggior parte degli stati quantistici non sono nè ortogonali nè identici, arriviamo ad una contraddizione. Allora il teorema del no-cloning è vero e non esiste un operatore universale unitario U tale che possa clonare uno stato quantistico sconosciuto. □

3.5.3 Piano dell'intercettatore

Per evitare che venga scoperto, Amos deve seguire un piano. Deve inviare una serie di qubit "sostituti" da mandare a Paolo. Essendo che non ha idea di quali basi abbia usato, anche lui deve generare una serie di basi casuali. Per semplicità assumiamo che Amos usi le stesse basi che ha usato per intercettare i qubit anche per codificare i bit di esca.

```
CANALE_QUANTISTICO = encode(amos_bits, amos_bases) # Invia i qubit
    a Paolo
```

Poi Paolo riceve i qubit e li misura nella base che ha scelto. Non sa che Amos li ha intercettati. Dopodiché Fabio deve comunicare a Paolo che basi ha usato per codificare i qubit. Gliele può comunicare attraverso qualsiasi canale di comunicazione classico. Essendo questo canale pubblico anche Amos saprà le basi che Paolo ha usato.

```
CANALE_CLASSICO = fabio_bases
```

Come al solito, Paolo controlla quali basi ha in comune con Fabio. Scarta quelle che non coincidono e mantiene quelle in comune. Successivamente comunica a Fabio le basi che ha in comune con lui.

Risultato finale

Essendo che Paolo e Fabio hanno ora solo i qubit misurati con le basi in comune, dovrebbero possedere gli stessi bit. Per fare sì che questo sia il caso, Fabio annuncerà i primi 100 bit che ha e Paolo dovrebbe avere gli stessi.

```
CANALE_CLASSICO = fabio_bits[:100]

# Paolo controlla se i primi 100 bit di Fabio corrispondono con i
    suoi primi 100 bit
if CANALE_CLASSICO == paolo_bits[:100]:
    print("Paolo e Fabio sembrano avere gli stessi bit.")
else:
    print("Almeno un bit e' risultato diverso.")
```

```
Almeno un bit e' risultato diverso.
```

Dopo aver confrontato i primi 100 bit, Fabio e Paolo si rendono conto che i loro bit non corrispondono. Assumendo che non ci sia nessun rumore nel canale quantistico o nei computer quantistici, Fabio e Paolo sono sicuri che il loro messaggio è stato intercettato. Di conseguenza decidono di non utilizzare la chiave e di ripetere il protocollo. La volta successiva potrebbero provare a utilizzare un canale di comunicazione quantistico diverso, così da confondere Amos.

Riferimenti bibliografici

[1] Qiskit Textbook [Internet]. GitHub; 1 Mar. 2025. Disponibile su: https://github.com/Qiskit/textbook/blob/main/notebooks/ch-algorithms/quantum-key-distribution.ipynb