UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA

DIPARTIMENTO DI FISICA "GIUSEPPE OCCHIALINI"

CORSO DI LAUREA TRIENNALE IN
FISICA

Tesi di laurea

Esecuzione e analisi del protocollo E91 per Quantum Key Distribution su computer quantistici

Relatore:
Dr. Andrea Giachero

Correlatore:
Dr. Danilo Labranca

Tesi di Laurea di:
Loris Coppa
Matricola 826237

Anno accademico:
2021/2022

Un computer che potrà calcolare
la Domanda alla Risposta Fondamentale,
un computer di tale infinita
e raffinata complessità che la stessa vita organica farà
parte della sua matrice operativa.
Progetterò questo computer per voi. E si chiamerà... Terra.

Indice

Introduzione

I computer quantistici sfruttano le proprietà della meccanica quantistica per ottenere risultati a cui i computer classici non possono arrivare.
Sono in grado di risolvere determinati problemi in tempi esponenzialmente minori rispetto ai dispositivi di calcolo classici; alcune delle applicazioni sono il machine learning, la simulazione di problemi fisici, la crittografia sicura, ecc.
Questo lavoro di tesi si occupa proprio di crittografia: si parla di "Quantum Key Distribution" (QKD) perché sono stati ideati diversi protocolli per comunicare una chiave privata in maniera sicura. Nella crittografia simmetrica, una stessa chiave viene usata per cifrare e per poi decifrare un messaggio, e la sua segretezza è quindi cruciale. L'avvento dei computer quantistici permette la comunicazione sicura di una chiave privata tra mittente e destinatario, impedendo a chiunque altro di entrarne in possesso. Questo era impossibile utilizzando informazioni classiche (non-quantistiche) in quanto teoricamente è possibile avere accesso a un canale di comunicazione classico e leggere le informazioni che vi passano attraverso.
Attualmente, infatti, i dispositivi che utilizziamo quotidianamente sfruttano metodi di crittografia asimmetrica, dove il problema della sicurezza della chiave privata non esiste. Nella crittografia asimmetrica un messaggio viene criptato con una chiave pubblica e decifrato con una chiave privata la quale è a disposizione solo del destinatario.
Metodi di questo tipo si basano su computazioni impossibili (o meglio, troppo complesse per essere risolte in tempi utili). Il sistema di crittografia asimmetrica RSA[1] utilizza come chiave pubblica un numero intero e come chiave privata i suoi fattori primi. Con un numero intero molto grande, la fattorizzazione è molto complicata, ed è dunque quasi impossibile ottenere la chiave privata con algoritmi classici.
Ma sfortunatamente uno dei compiti in cui eccelle il computer quantistico è la fattorizzazione in numeri primi, e questo rende potenzialmente vulnerabili gli attuali sistemi standard di crittografia.
Bisogna quindi passare a sistemi di crittografia che non siano violabili neanche da computer quantistici, come i protocolli di QKD; uno di questi è il protocollo E91[2], il quale è il soggetto di questa tesi. Esso si basa sullo scambio di qubit in entanglement. La misura indesiderata di uno di questi qubit fa collassare lo stato quantistico, il quale attraverso un test statistico rivela l'intromissione da parte di terzi.

---

[1] Dai nomi dei progettisti Ronald Rivest, Adi Shamir e Leonard Adleman

[2] Introdotto da Artur Ekert nel 1991

Capitolo 1

Computer quantistici e protocollo E91

1.1 Qubit e gate

La computazione quantistica è un campo di ricerca che mira a sfruttare le proprietà uniche della meccanica quantistica per eseguire calcoli e risolvere problemi irrisolvibili su elaboratori classici. I computer quantistici operano utilizzando qubit che possono rappresentare 0 o 1, ma anche qualsiasi combinazione lineare di questi stati. Questa proprietà, nota come principio di sovrapposizione (superposition), consente ai computer quantistici di eseguire determinati calcoli molto più velocemente dei computer classici (per esempio, l'algoritmo di Grover per la ricerca di un elemento in una lista[5]).
Il qubit, quindi, è l'unità fondamentale d'informazione gestita dai computer quantistici, ed è rappresentato da un sistema quantistico a due stati. Il suo nome deriva da "quantum bit", in analogia con i bit utilizzati dai computer classici che possono valere soltanto 0 o 1.

Matematicamente, un qubit può essere rappresentato da un vettore unitario in uno spazio di Hilbert complesso a due dimensioni, noto come vettore di stato:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{1.1}$$

dove $|\psi\rangle$ è il vettore di stato, $|0\rangle$ e $|1\rangle$ sono i vettori della base canonica e α e β sono coefficienti complessi. Quando un qubit viene osservato, il modulo quadro dei coefficienti α e β rappresenta la probabilità di ottenere 0 o 1, rispettivamente, come risultato della misura.

Lo stato di un qubit può essere manipolato attraverso l'applicazione di matrici unitarie quadrate di dimensione 2, note come "porte quantistiche" (gate) in analogia con le porte logiche della computazione classica.
Il motivo per cui sia i vettori di stato che i gate debbano essere unitari è dovuto al fatto che la probabilità totale deve sempre essere 1, cioè che $|\alpha|^2 + |\beta|^2 = 1$

1.2 Crittografia simmetrica e asimmetrica

Per capire l'utilità delle tecniche di Quantum Key Distribution (QKD) e del protocollo E91[1] bisogna conoscere i principi alla base dei protocolli di crittografia comunemente usati in informatica per la trasmissione di messaggi sicuri.

È consuetidine chiamare A e B i protagonisti di una comunicazione, cioé mittente e destinatario, oppure "Alice" e "Bob". Una terza parte che cerca di svelare il contenuto del messaggio comunicato viene di solito indicato con la lettera E, oppure "Eve" (dall'inglese "eavesdropper", che significa intercettatore).

I vari protocolli esistenti si dividono principalmente in protocolli a crittografia simmetrica e asimmetrica.
La crittografia simmetrica, nota anche come crittografia a chiave privata, è un tipo di metodo crittografico che utilizza la stessa chiave sia per cifrare che per decifrare un messaggio. In un sistema di crittografia simmetrica, Alice e Bob condividono in anticipo una chiave segreta, e Alice utilizza la chiave per cifrare un messaggio prima di inviarlo a Bob. Bob può quindi utilizzare la stessa chiave per decifrare il messaggio e leggerne il contenuto.

In particolare, utilizzando un sistema crittografico OTP (One-Time Pad) si può ottenere un messaggio impossibile da decriptare per chi non è in possesso della chiave.
La chiave di una tecnica OTP deve soddisfare i seguenti quattro requisiti per essere completamente inviolabile:

1. Deve essere lunga almeno quanto il messaggio da cifrare;
2. Deve essere completamente casuale;
3. Non deve mai essere riutilizzata in tutto o in parte;
4. Deve essere tenuta completamente segreta tra il mittente e il destinatario.

Il problema principale di un sistema OTP è quindi la condivisione della chiave. Siccome deve cambiare a ogni trasmissione, Alice e Bob dovranno comunicare in qualche modo la chiave segreta tramite un canale di comunicazione che non può mai essere sicuro al 100%.

La crittografia asimmetrica, invece, è un tipo di metodo crittografico che utilizza una coppia di chiavi, una chiave pubblica e una chiave privata, utilizzabili per cifrare e per decifrare un messaggio. In un sistema di crittografia asimmetrica, Alice e Bob possiedono una chiave pubblica e una chiave privata a testa. Alice può utilizzare la chiave pubblica di Bob per cifrare un messaggio

[1] Introdotto da Artur Ekert nel 1991

e inviarlo a Bob, e Bob può utilizzare la propria chiave privata per decifrare il messaggio e leggerne il contenuto. Allo stesso modo, Bob può utilizzare la chiave pubblica di Alice per cifrare un messaggio e inviarlo ad Alice, e lei può utilizzare la propria chiave privata per decifrare il messaggio.

Il principale vantaggio della crittografia asimmetrica è che permette una comunicazione sicura tra le parti senza che debbano condividere una chiave segreta. Lo svantaggio di questo meccanismo è che per poter funzionare, la chiave pubblica e la chiave privata appartenenti a un soggetto hanno una correlazione matematica, e dunque si può in teoria risalire alla chiave privata conoscendo quella pubblica (che è appunto di dominio pubblico). I protocolli più utilizzati usano come chiave pubblica un numero intero molto grande, e come chiave privata la sua fattorizzazione in numeri primi; la sicurezza è garantita dal fatto che, mentre il calcolo del prodotto di numeri primi è computazionalmente molto semplice, il contrario (cioè il calcolo della fattorizzazione di un numero) è molto complicato se il numero di partenza è molto grande. Quindi è praticamente impossibile scoprire la chiave privata che ha generato la chiave pubblica.

La computazione quantistica cambia le carte in tavola per entrambe le tipologie. Da un lato, permette di implementare algoritmi che calcolano la fattorizzazione in numeri primi in tempi molto più rapidi rispetto ai computer classici, rendendo quindi più vulnerabili gli attuali protocoli di crittografia asimmetrica. Dall'altro lato, però, permette di implementare algoritmi di trasmissione di chiave completamente sicuri (Quantum Key Distribution), tra cui il protocollo E91.

1.3 Protocollo E91

Artur K. Ekert nel 1991 (da qui il nome E91) ha ideato un protocollo per la distribuzione di chiavi private che utilizza coppie di qubit in entanglement e il teorema di Bell.[4]
Due o più sistemi fisici (particelle o qubit) si dicono in entanglement se insieme formano un nuovo sistema il cui stato è una combinazione degli stati dei singoli sistemi separati. Una misura effettuata su uno dei sotto-sistemi determina il valore della stessa osservabile sugli altri.
Il teorema di Bell conferma l'esistenza dell'entanglement attraverso delle disuguaglianze: una teoria quantistica locale, che non ammette quindi l'esistenza dell'entanglement, prevede che le disuguaglianze di Bell siano rispettate. Diversi esperimenti hanno invece misurato una violazione di queste disuguaglianze.[7]

Il funzionamento del protocollo E91 è il seguente: Alice vuole inviare dei dati a Bob; Alice genera una coppia di qubit in entanglement e ne invia uno a Bob. In un caso reale potrà essere conveniente far generare la coppia di qubit

a un ente terzo "Charlie" che distribuirà i qubit ad Alice e Bob.
Lo stato in entanglement che viene utilizzato in questo caso è quello di singoletto, cioè:

$$|\psi\rangle = \frac{|0\rangle_A \otimes |1\rangle_B - |1\rangle_A \otimes |0\rangle_B}{\sqrt{2}} = \frac{|01\rangle - |10\rangle}{\sqrt{2}} \tag{1.2}$$

Dove $\otimes$ è l'operatore di prodotto tensoriale. $|0\rangle_A \otimes |1\rangle_B$, o più semplicemente $|01\rangle$, è un vettore di uno spazio di Hilbert dato dal prodotto tensoriale dello spazio di Hilbert dei due qubit, che sarà quindi di dimensione 4: $H = H_A \otimes H_B$

I qubit saranno in genere fotoni o particelle di spin 1/2 e l'osservabile sarà la polarizzazione del fotone o il valore dello spin in una certa direzione.
Le coppie di qubit condivise in questo modo saranno N, e Alice e Bob misureranno ogni qubit ricevuto in una direzione casuale scelta tra 3 direzioni prestabilite. Per Alice gli angoli saranno:

$$\phi_1 = 0; \quad \phi_2 = \frac{\pi}{4}; \quad \phi_3 = \frac{\pi}{2} \tag{1.3}$$

Per Bob gli angoli saranno:

$$\theta_1 = \frac{\pi}{4}; \quad \theta_2 = \frac{\pi}{2}; \quad \theta_3 = \frac{3}{4}\pi \tag{1.4}$$

Questo significa che verrà misurata la polarizzazione del fotone (o lo spin della particella) lungo la direzione indicata da questi 6 angoli, a cui corrispondono quindi degli operatori, detti gate, che chiameremo rispettivamente $A_1, A_2, A_3, B_1, B_2, B_3$.
Dopo l'applicazione di questi gate, Alice e Bob misurano i loro rispettivi qubit e possono ottenere come risultato 0 o 1, a cui assegnano il valore di -1 e +1, rispettivamente (corrisponde al valore misurato dello spin in unità di $\hbar/2$).
Siamo interessati a calcolare il valore:

$$S = \langle A_1 B_1 \rangle - \langle A_1 B_3 \rangle + \langle A_3 B_1 \rangle + \langle A_3 B_3 \rangle \tag{1.5}$$

Dove ci aspettiamo un valore teorico di $S = -2\sqrt{2}$ (limite di Tsirelson[2])
Invece, per le coppie misurate lungo la stessa direzione ci aspettiamo un valore anti-correlato:

$$\langle A_2 B_1 \rangle = \langle A_3 B_2 \rangle = -1 \tag{1.6}$$

Il protocollo quindi sfrutta i qubit misurati lungo la stessa direzione (A_2 e B_1; A_3 e B_2) come chiave segreta, mentre tutti gli altri vengono utilizzati per calcolare il valore S.

Dopo la misurazione, Alice e Bob comunicano su un canale non-sicuro le direzioni che hanno usato per misurare ogni qubit. A questo punto è noto

pubblicamente quali sono i qubit utilizzati come chiave; per questi, Bob saprà che per ogni ± 1 misurato corrisponde un ∓ 1 della chiave inviata da Alice. Se Eve tenta di interferire in qualche modo, il valore di S calcolato tornerà al limite non-quantistico, cioè quello dettato dalla disuguaglianza CHSH (da Clauser-Horne-Shimony-Holt[3], una delle disuguaglianze di Bell):

$$|S| \leq 2 \tag{1.7}$$

Quindi, se Alice e Bob trovano un valore di $|S|$ minore di 2 sapranno che la comunicazione non è sicura e dovranno usare un altro canale di comunicazione. Si noti che, comunque sia, Eve ha accesso solo ai qubit della chiave, e non al contenuto del messaggio in sè: la chiave viene generata casualmente a ogni comunicazione e dunque Eve non potrà mai leggere i dati in chiaro con questo metodo.

Riassumendo, il protocollo avviene seguendo questi passi:

1. Alice decide di inviare dei dati a Bob;
2. Alice chiede a Charlie di generare N coppie di qubit in entanglement;
3. Charlie invia i qubit di ogni coppia ad Alice e Bob;
4. Sia Alice che Bob misurano i propri qubit lungo una delle direzioni prestabilite, selezionandole casualmente;
5. Alice e Bob memorizzano i risultati delle loro misure, in ordine di arrivo dei qubit, e comunicano pubblicamente la direzione usata per ogni misurazione;
6. I qubit da utilizzare come chiave sono quindi noti. I risultati delle misurazioni di tutti i qubit non-chiave vengono invece comunicati pubblicamente;
7. Alice e Bob possono quindi calcolare il valore di S;
8. Se $|S| > 2$ la comunicazione è sicura: Alice può usare la chiave condivisa con Bob per cifrare il suo messaggio e procedere a inviarlo a Bob tramite un qualsiasi canale;
9. Se $|S| \leq 2$ la comunicazione non è sicura: Eve è intervenuta e ha cercato di ottenere la chiave; Alice e Bob dovranno quindi utilizzare un altro canale di comunicazione e ripetere il protocollo dall'inizio.

Capitolo 2

Implementazione del protocollo E91 su ambiente IBM

Operativamente, è possibile testare il protocollo E91 su computer quantistici reali e su simulatori.
Per questo lavoro di tesi si è scelto di usare IBM Quantum Lab (in precedenza IBM Quantum Experience) che mette a disposizione simulatori e computer quantistici reali con i quali è possibile interagire attraverso Qiskit, il quale è un software development kit (SDK) open-source.
Verrà presentato in questo capitolo un programma Python che, tramite Qiskit, simula il comportamento di Alice, Bob e Eve in una comunicazione tramite protocollo E91.

2.1 Qiskit

Qiskit è un pacchetto di librerie sviluppato nel linguaggio Python che permette la costruzione dei circuiti quantistici a livello logico. Questi circuiti vengono costruiti in oggetti appositi tramite le classi messe a disposizione dalle librerie; le stesse permettono poi di eseguire questi circuiti su diversi simulatori ma soprattutto permettono di eseguirli anche su computer quantistici reali messi a disposizione online da IBM.
I computer quantistici di IBM sono vari e ognuno ha caratteristiche diverse; i computer a cui abbiamo accesso hanno tutti 5 o 7 qubit, ma il numero di qubit non è l'unica proprietà importante.
Nel capitolo 4 verranno analizzate le differenze tra i vari computer e verranno illustrati i risultati del programma ottenuti da due di questi.
La versione di Qiskit utilizzata per questo lavoro è la 0.36.1[6].

2.2 Gate necessari

Per implementare il protocollo E91 con Qiskit vengono utilizzati diversi gate.

Il gate X agisce come un NOT sugli stati $|0\rangle$ e $|1\rangle$:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$X|0\rangle = |1\rangle$$
$$X|1\rangle = |0\rangle$$

Il gate H (Hadamard) permette di portare un qubit in uno stato di sovrapposizione:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \equiv |+\rangle$$

$$H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \equiv |-\rangle$$

$$H|+\rangle = |0\rangle$$
$$H|-\rangle = |1\rangle$$

Il gate $CNOT$ (Controlled NOT) agisce su due qubit e applica un gate X a un qubit in base al valore di un altro qubit (qubit di controllo). Il qubit di controllo quindi non cambia il suo valore dopo l'applicazione di un $CNOT$. Se il primo qubit è quello di controllo, il gate $CNOT$ agisce nel seguente modo:

$$CNOT|00\rangle = |00\rangle$$
$$CNOT|01\rangle = |01\rangle$$
$$CNOT|10\rangle = |11\rangle$$
$$CNOT|11\rangle = |10\rangle$$

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

I gate H e $CNOT$ combinati assieme permettono di creare uno stato di entanglement tra i due qubit.

Per le misurazioni di Alice e Bob verranno inoltre usati i gate S, T, $T^\dagger$, che aggiungono una differenza di fase tra la componente $|0\rangle$ e la componente $|1\rangle$ di un qubit:

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$$

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$$

$$T^\dagger = \begin{pmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{pmatrix}$$

2.3 Implementazione del protocollo E91

Di seguito verranno elencate le varie parti che compongono il programma.
Il codice completo è riportato nell'appendice A.

2.3.1 Generazione delle coppie di qubit

Per generare una singola coppia di qubit in stato di singoletto viene usata una funzione helper, `newEntangledCircuit()`.

```
# Lo stato inziale sarà |01> - |10> / sqrt(2)
def newEntangledCircuit():
    newCircuit = QuantumCircuit(3, 3) #Il terzo qubit serve per
    ↪  implementare la strategia 3 di Eve
    newCircuit.x(0)
    newCircuit.x(1)
    newCircuit.h(0)
    newCircuit.cnot(0, 1)
    return newCircuit
```

La classe `QuantumCircuit`, dalle librerie di Qiskit, rappresenta un circuito di qubit, gate e operazioni di misura sui qubit. Viene inizializzato in questo caso con 3 registri quantistici (cioè 3 qubit) e 3 registri classici (cioè 3 bit, dove verranno registrati i risultati delle misure sui 3 qubit).
Il primo qubit corrisponde a quello di Alice, il secondo a quello di Bob, il terzo a quello che Eve può utilizzare nelle sue strategie.
In Qiskit tutti i qubit vengono inizializzati nello stato $|0\rangle$, pertanto vanno portati nello stato $|1\rangle$ tramite l'applicazione di una porta NOT (gate X), dopodiché vengono messi in entanglement tramite i gate H e $CNOT$ che portano infine al risultato desiderato.

Il circuito generato è quello mostrato in Figura 2.1:

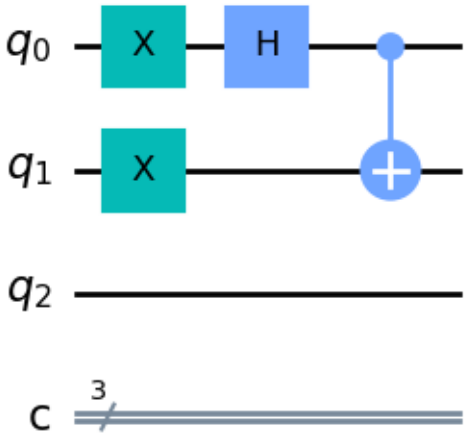

Figura 2.1: Circuito per generare uno stato di singoletto per due qubit

Dove q_0 è il qubit di Alice, q_1 è il qubit di Bob, q_2 è il qubit di Eve (che non viene utilizzato inizialmente), mentre c rappresenta i 3 registri classici in cui vengono salvati i risultati delle misure dei 3 qubit sottoforma di bit classici.

Con un parametro `n` viene deciso innanzitutto quante coppie di qubit creare. Nel corpo principale del programma vengono quindi definite le prime variabili utili al protocollo:

```
n=100

alice_basis = []
bob_basis = []

qubits_circuits = []

for i in range(0, n):
    #Charlie (o Alice) genera n coppie di qubit in entanglement
    qubits_circuits.append(newEntangledCircuit())

    #Alice e Bob scelgono indipendentemente una delle 3 basi
    ↪  previste per loro con cui misurare i propri qubit
    alice_basis.append(random.choice([1, 2, 3]))
    bob_basis.append(random.choice([1, 2, 3]))
```

In particolare, l'array `qubits_circuits` conterrà tutti i circuiti generati dalla funzione `newEntangledCircuit()` che viene chiamata n volte dal ciclo for. Per semplicità, nello stesso ciclo vengono anche decise le basi che Alice e Bob useranno per la misura dei qubit.

2.3.2 Misura dei qubit

Si dà per scontato che Alice e Bob a questo punto ricevano i qubit che spettano loro.
Scegliendo un numero casuale tra 1 e 3, Alice e Bob scelgono quale base usare per la misura dei qubit: a seconda della base scelta, verranno applicati dei gate diversi.
A_1, A_2, A_3, B_1, B_2 e B_3 vengono trattati come dei gate composti da altri gate, applicati al circuito tramite una funzione apposita:

```
# X gate
def A1(circ: QuantumCircuit):
    circ.h(0)

# Corrisponde a Z + X / sqrt(2)
def A2(circ: QuantumCircuit):
    circ.s(0)
    circ.h(0)
    circ.t(0)
    circ.h(0)

# Z gate
def A3(circ: QuantumCircuit):
    return #Nessuna operazione richiesta per la direzione Z


# Corrisponde a Z + X / sqrt(2)
def B1(circ: QuantumCircuit):
    circ.s(1)
    circ.h(1)
    circ.t(1)
    circ.h(1)

# Z
def B2(circ: QuantumCircuit):
    return  # Nessuna operazione richiesta per la direzione Z

# Corrisponde a Z - X / sqrt(2)
def B3(circ: QuantumCircuit):
    circ.s(1)
    circ.h(1)
    circ.tdg(1)
    circ.h(1)
```

Di seguito i circuiti corrispondenti ai gate A_1, A_2, A_3, B_1, B_2 e B_3:

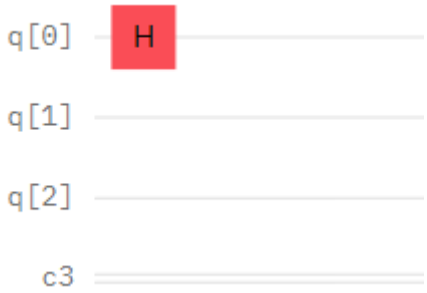

Figura 2.2: Circuito che rappresenta A_1

Figura 2.3: Circuito che rappresenta A_2

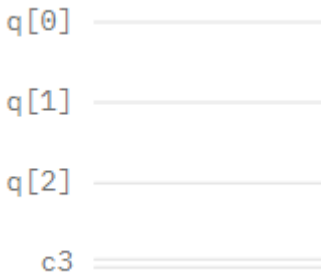

Figura 2.4: Circuito che rappresenta A_3

Figura 2.5: Circuito che rappresenta B_1

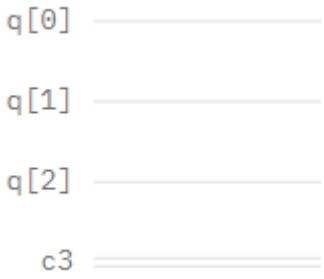

Figura 2.6: Circuito che rappresenta B_2

Figura 2.7: Circuito che rappresenta B_3

Si noti che nelle Figure 2.4 e 2.6 il gate è rappresentato da un circuito vuoto perché non bisogna effettuare nessuna operazione per misurare un qubit lungo l'asse Z in Qiskit.

Ricordando gli angoli della direzione di misura per Alice:

$$\phi_1 = 0; \quad \phi_2 = \frac{\pi}{4}; \quad \phi_3 = \frac{\pi}{2} \tag{2.1}$$

e per Bob:

$$\theta_1 = \frac{\pi}{4}; \quad \theta_2 = \frac{\pi}{2}; \quad \theta_3 = \frac{3}{4}\pi \tag{2.2}$$

Bisogna tenere conto che Qiskit effettua misure soltanto lungo l'asse Z, cioè nella base canonica $|0\rangle, |1\rangle$. Questo implica che occorre ruotare la base lungo la direzione in cui vogliamo effettuare la misura.
La direzione a 90° corrisponde già all'asse Z; per la direzione a 0° è sufficiente l'applicazione di un gate H. Per le direzioni a 45° e 135° bisogna utilizzare anche i gate T ed S che effettuano le rotazioni necessarie.

Per misurare e ottenere il risultato dei qubit vengono usate due ulteriori funzioni d'appoggio:

- `getResultsFromCircuits` riceve come input la lista dei circuiti dei qubit da eseguire e il backend su cui eseguirli. Restituisce il risultato del job eseguito sul backend, che contiene i risultati delle misure di ogni circuito;

- `indexForResult` converte il risultato in un indice da 0 a 3. Risulta conveniente per contare i diversi risultati.

```
def getResultsFromCircuits(circuits: List[QuantumCircuit], backend:
↪  Backend):
    job = execute(circuits, backend, shots=1)
    print(job.status())
    results = job.result()
    print(job.status())
    print("Data e ora Job: ", job.creation_date())
    print("Data e ora completamento: ", datetime.now())
    return results

def indexForResult(result: str) -> int:
    result = result[1:3] #I risultati sono al contrario, il terzo bit
    ↪  quindi è quello di Alice, il secondo quello di Bob, il terzo
    ↪  quello di Eve
    if result == '00':
        return 0
    elif result == '01':
        return 1
    elif result == '10':
        return 2
    elif result == '11':
        return 3
```

Sfruttando queste funzioni, nel corpo principale del programma vengono applicati tutti i gate scelti e vengono conseguentemente misurati tutti i qubit. I conteggi dei risultati di queste misure vengono salvati in 16 variabili (4 array da 4 elementi):

```
#Conteggi per 00, 01, 10, 11, rispettivamente
countA1B1 = [0, 0, 0, 0] # XW observable
countA1B3 = [0, 0, 0, 0] # XV observable
countA3B1 = [0, 0, 0, 0] # ZW observable
countA3B3 = [0, 0, 0, 0] # ZV observable


aliceKey = []
bobKey = []

for i, circ in enumerate(qubits_circuits):
    if alice_basis[i] == 3 and bob_basis[i] == 2: #Verranno usati per
    ↪  la chiave
        A3(circ)
        B2(circ)
    elif alice_basis[i] == 2 and bob_basis[i] == 1: #Verranno usati
    ↪  per la chiave
        A2(circ)
```

```
            B1(circ)
        elif alice_basis[i] == 1 and bob_basis[i] == 3:
            A1(circ)
            B3(circ)
        elif alice_basis[i] == 1 and bob_basis[i] == 1:
            A1(circ)
            B1(circ)
        elif alice_basis[i] == 3 and bob_basis[i] == 1:
            A3(circ)
            B1(circ)
        elif alice_basis[i] == 3 and bob_basis[i] == 3:
            A3(circ)
            B3(circ)

        circ.measure(0, 0)
        circ.measure(1, 1)
        circ.measure(2, 2)


backendResults = getResultsFromCircuits(qubits_circuits, _backend)

for i, result in enumerate(backendResults.get_counts()):
    resultBits = list(result.keys())[0]
    if (alice_basis[i] == 3 and bob_basis[i] == 2) or (alice_basis[i]
    ↪  == 2 and bob_basis[i] == 1):
        aliceKey.append(resultBits[2])
        bobKey.append(resultBits[1])
    elif alice_basis[i] == 1 and bob_basis[i] == 3:
        j = indexForResult(resultBits)
        countA1B3[j] += 1
    elif alice_basis[i] == 1 and bob_basis[i] == 1:
        j = indexForResult(resultBits)
        countA1B1[j] += 1
    elif alice_basis[i] == 3 and bob_basis[i] == 1:
        j = indexForResult(resultBits)
        countA3B1[j] += 1
    elif alice_basis[i] == 3 and bob_basis[i] == 3:
        j = indexForResult(resultBits)
        countA3B3[j] += 1
```

In tutto, le combinazioni possibili di gate applicati da Alice e Bob generano 9 circuiti diversi, mostrati qui di seguito:

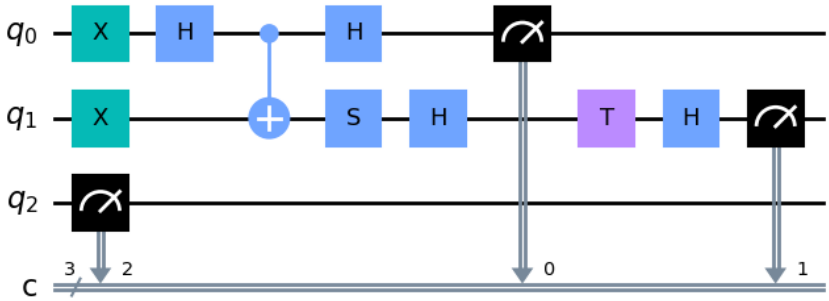

Figura 2.8: Circuito generato dall'applicazione di A_1 e B_1 ai qubit in entanglement

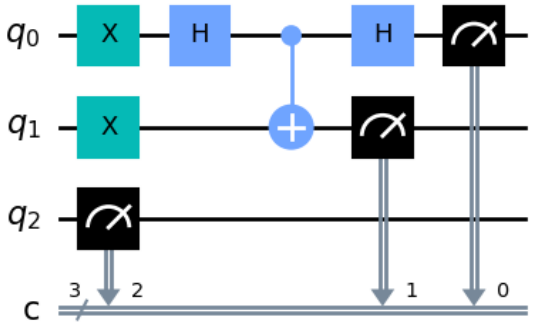

Figura 2.9: Circuito generato dall'applicazione di A_1 e B_2 ai qubit in entanglement

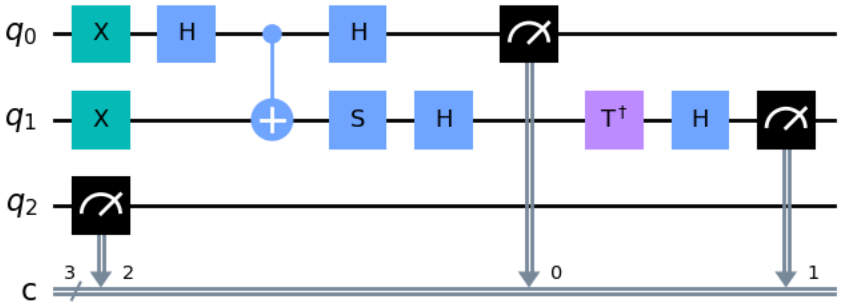

Figura 2.10: Circuito generato dall'applicazione di A_1 e B_3 ai qubit in entanglement

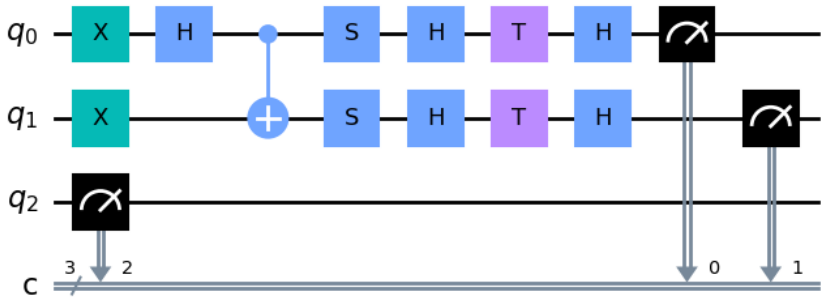

Figura 2.11: Circuito generato dall'applicazione di A_2 e B_1 ai qubit in entanglement

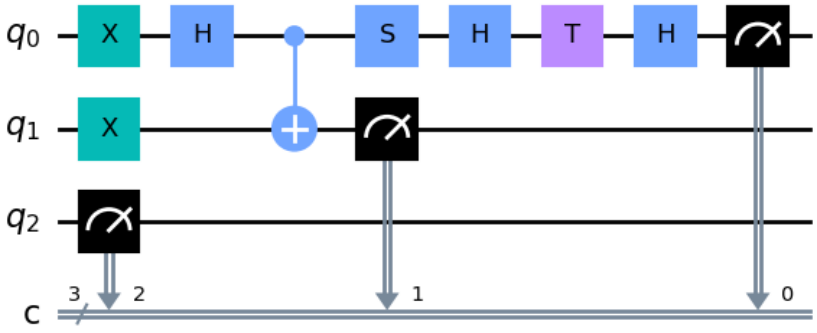

Figura 2.12: Circuito generato dall'applicazione di A_2 e B_2 ai qubit in entanglement

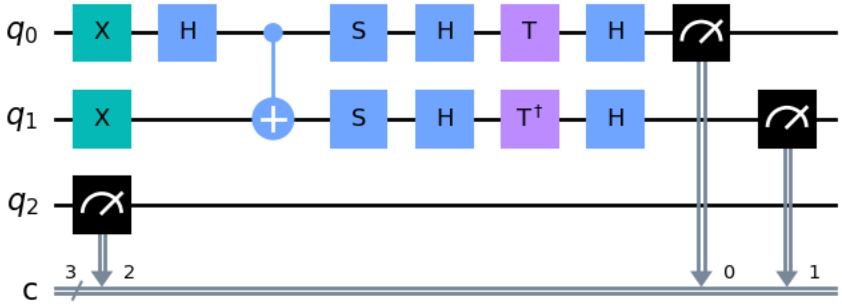

Figura 2.13: Circuito generato dall'applicazione di A_2 e B_3 ai qubit in entanglement

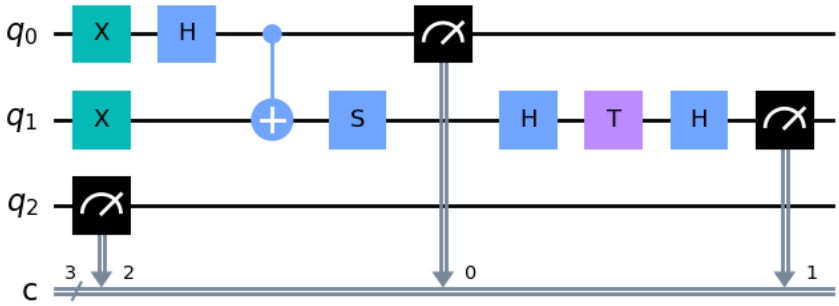

Figura 2.14: Circuito generato dall'applicazione di A_3 e B_1 ai qubit in entanglement

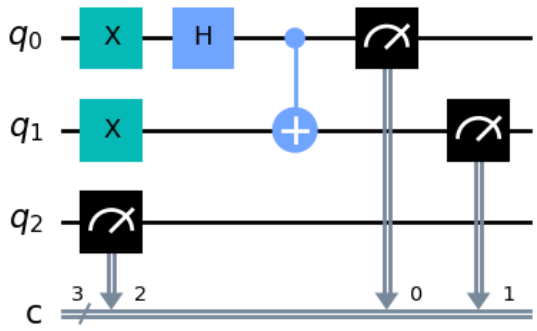

Figura 2.15: Circuito generato dall'applicazione di A_3 e B_2 ai qubit in entanglement

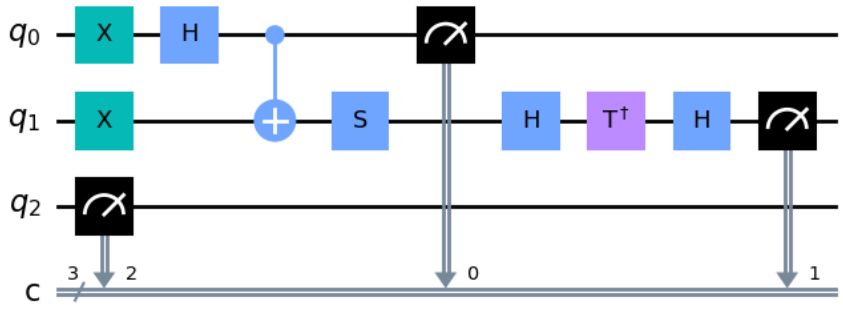

Figura 2.16: Circuito generato dall'applicazione di A_3 e B_3 ai qubit in entanglement

2.3.3 Calcolo del valore di correlazione S

Infine, occorre calcolare il valore di correlazione S che permette ad Alice e Bob di sapere se la comunicazione della chiave è avvenuta in maniera indisturbata.

Ricordando la disuguaglianza CHSH:

$$|S| \leq 2 \tag{2.3}$$

Dove:

$$S = \langle A_1 B_1 \rangle - \langle A_1 B_3 \rangle + \langle A_3 B_1 \rangle + \langle A_3 B_3 \rangle \tag{2.4}$$

Ricaviamo il valore di aspettazione:

$$E(A_i, B_j) = \langle A_i B_j \rangle = \sum_k V_k \; P_k(A_i, B_j) \tag{2.5}$$

Cioè la somma su tutti i risultati possibili (indice k), dove V_k è il valore delle due misure dei qubit moltiplicate tra loro e $P_k(A_i, B_j)$ è la probabilità di ottenere quel risultato con le direzioni scelte da Alice e Bob.
Quindi, V_k varrà +1 per il risultato 00, -1 per 01, -1 per 10, e +1 per 11.
Dunque:

$$E(A_i, B_j) = P_{00}(A_i, B_j) + P_{11}(A_i, B_j) - P_{01}(A_i, B_j) - P_{10}(A_i, B_j) \tag{2.6}$$

Il programma quindi calcola queste probabilità dai conteggi dei risultati ottenuti dalle misure:

$$P_k(A_i, B_j) = \frac{N_k(A_i, B_j)}{N_{tot}(A_i, B_j)} \tag{2.7}$$

Dove $N_k(A_i, B_j)$ è il numero di risultati k ottenuti utilizzando le basi A_i e B_j, e $N_{tot}(A_i, B_j)$ è il totale di misure effettuate in quelle basi.

Il codice è quindi il seguente:

```
total11 = sum(countA1B1)
total13 = sum(countA1B3)
total31 = sum(countA3B1)
total33 = sum(countA3B3)

# Valori d'aspettazione
expect11 = (countA1B1[0] - countA1B1[1] - countA1B1[2] +
↪  countA1B1[3]) / total11  # -1/sqrt(2)
expect13 = (countA1B3[0] - countA1B3[1] - countA1B3[2] +
↪  countA1B3[3]) / total13  # 1/sqrt(2)
expect31 = (countA3B1[0] - countA3B1[1] - countA3B1[2] +
↪  countA3B1[3]) / total31  # -1/sqrt(2)
expect33 = (countA3B3[0] - countA3B3[1] - countA3B3[2] +
↪  countA3B3[3]) / total33  # -1/sqrt(2)

# Valore di correlazione CHSH
corr = expect11 - expect13 + expect31 + expect33
```

2.3.4 Controllo errori nella chiave

In un computer quantistico reale, al contrario di quanto avviene sul simulatore, alcune misure possono essere affette da errori, restituendo un bit opposto a quello che ci si aspetta. Pertanto è conveniente confrontare le chiavi di Alice e Bob per contare eventuali differenze:

```
aliceKeyString = ""
bobKeyString = ""
errorCount = 0

for (i, aliceBitString) in enumerate(aliceKey):
    bobBitString = bobKey[i]
    bobBitString = str(int(not bool(int(bobBitString)))) #applico NOT
    aliceKeyString += aliceBitString
    bobKeyString += bobBitString
    if aliceBitString != bobBitString:
        errorCount += 1
```

Non solo l'imprecisione di un computer reale può incidere sulla correttezza della chiave, ma anche l'intervento esterno di Eve.
Le possibili strategie che Eve può adottare per ottenere la chiave privata

verranno discusse nel capitolo 3, assieme all'analisi dei risultati sul simulatore.

Capitolo 3

Esecuzione del protocollo E91 su simulatore

3.1 Raccolta dati senza Eve

Per prima cosa il programma è stato eseguito sul simulatore senza l'azione di Eve, per osservare il comportamento del protocollo in condizioni ottimali. Il simulatore in questo caso è utile per capire quante coppie di qubit servono per ottenere un risultato utilizzabile.
I dati raccolti sono il valore S (in realtà $|S|$) calcolato per la disuguaglianza CHSH, il valore Δ (cioè la differenza tra S e il valore teorico di $2\sqrt{2} = 2,82842...$) e infine la lunghezza L della chiave generata con questo metodo.
Si noti che la chiave è composta da tutti i bit per i quali Alice e Bob hanno usato la stessa base di misura. Questo avviene solo in 2 casi su 9, pertanto la lunghezza media della chiave sarà:

$$\langle L \rangle = \frac{2}{9}\, n$$

Per ogni raccolta dati, il programma è stato eseguito 10 volte per considerare eventuali fluttuazioni statistiche.
La prima raccolta dati, nella tabella 3.1, è stata effettuata utilizzando $n = 100$, cioè 100 coppie di qubit in entanglement distribuite ad Alice e Bob:

S	Δ	L
3.21	0.38	22
2.74	-0.09	25
2.68	-0.14	23
3.31	0.48	22
2.89	0.06	16
3.04	0.21	30
2.58	-0.25	19
3.05	0.22	17
2.65	-0.17	25
3.00	0.18	21

Tabella 3.1: Dati raccolti col simulatore con 100 coppie di qubit

Il valore di S è molto variabile ma mai inferiore a 2.

La raccolta seguente è stata eseguita utilizzando 500 qubit, nella tabella 3.2:

S	Δ	L
2.87	0.04	111
2.64	-0.18	116
2.62	-0.21	100
2.65	-0.18	136
3.01	0.18	117
2.53	-0.30	113
2.95	0.12	104
2.79	-0.04	114
2.72	-0.11	130
2.97	0.14	103

Tabella 3.2: Dati raccolti col simulatore con 500 coppie di qubit

Qui il valore di S si avvicina di più al valore teorico di $2\sqrt{2} = 2.83$, con una media di 2.77 ± 0.16 rispetto alla media di 2.92 ± 0.23 per $n = 100$.

Sono state eseguite altre raccolte dati con $n = 1000$ e $n = 2000$, per verificare ulteriori miglioramenti. I dati sono riportati qui di seguito nelle Tabelle 3.3 e 3.4:

S	Δ	L
2.87	0.04	229
2.79	-0.04	214
2.59	-0.24	207
2.93	0.10	210
2.75	-0.08	224
2.96	0.13	226
2.68	-0.15	219
2.94	0.11	223
2.75	-0.08	216
2.61	-0.22	216

Tabella 3.3: Dati raccolti col simulatore con 1000 coppie di qubit

S	Δ	L
2.81	-0.02	466
2.89	0.06	444
2.79	-0.04	444
2.88	0.05	450
2.76	-0.07	463
2.79	-0.03	441
2.72	-0.11	460
2.79	-0.04	429
2.78	-0.05	451
2.77	-0.05	456

Tabella 3.4: Dati raccolti col simulatore con 2000 coppie di qubit

La crescente precisione del calcolo di S è ancora più evidente guardando il valore medio per ogni raccolta dati. Vengono riportati di seguito le medie μ, l'errore sulla media σ, la differenza tra media e valore teorico $|\Delta_\mu| = |\mu - 2\sqrt{2}|$ e infine la correzione dovuta all'errore, nel caso peggiore, cioè $|\Delta_\mu| + \sigma$:

n	μ	σ	$\lvert\Delta_\mu\rvert$	$\lvert\Delta_\mu\rvert + \sigma$
100	2.92	0.23	0.09	0.32
500	2.77	0.16	0.05	0.21
1000	2.79	0.13	0.04	0.17
2000	2.80	0.05	0.03	0.08

Tabella 3.5: Medie dei valori S sul simulatore

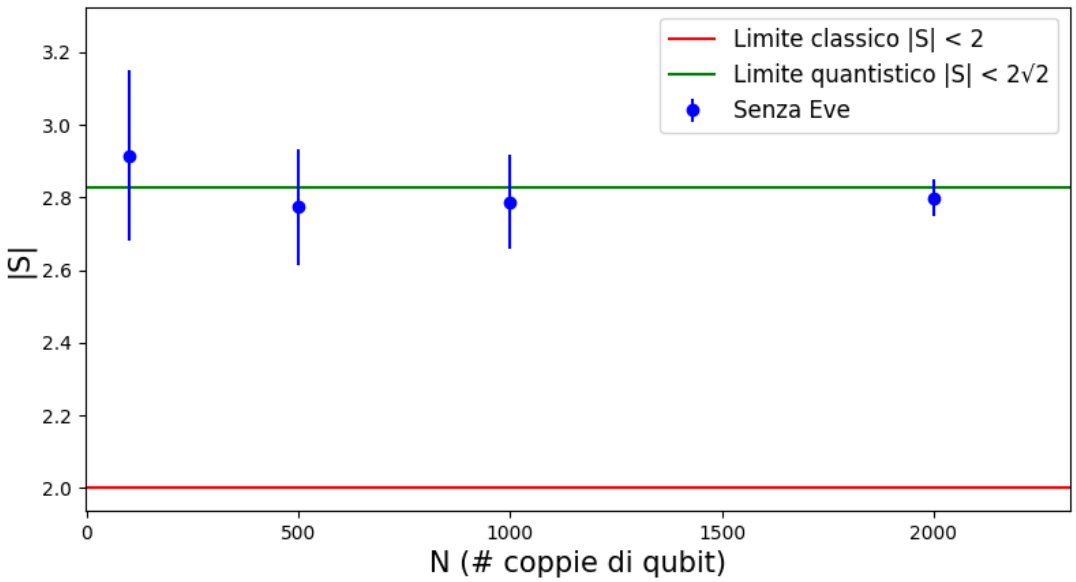

Figura 3.1: Confronto valori medi di S ottenuti sul simulatore

Il grafico in figura 3.1 mette in evidenza come i valori di S ottenuti, con il loro errore, sono compatibili con il valore teorico di $2\sqrt{2}$.

3.2 Raccolta dati con l'intervento di Eve

Viene ora verificato come un terzo ente, Eve, potrebbe appropriarsi della chiave privata e riuscire così a decifrare un messaggio criptato con la stessa. Per Eve, l'unico accesso ai qubit di Alice e Bob si trova tra il punto 3 e il punto 4 del protocollo, cioè durante la trasmissione dei qubit da Charlie ad Alice e Bob (vedasi sezione 1.3).
È importante premettere che Eve non può clonare nessun qubit: esiste infatti per i computer quantistici un "teorema di no-cloning" che vieta l'esistenza di un gate capace di clonare un generico qubit.[8]

Teorema 1 (No-cloning theorem)**.** *Non può esistere nessun gate che permette di creare una copia identica di un qubit arbitrario e sconosciuto.*

Dimostrazione. Sia $|\psi\rangle$ lo stato di un qualsiasi qubit A. Sia U un gate tale per cui:

$$U(|\psi\rangle_A \otimes |0\rangle_B) = |\psi\rangle_A \otimes |\psi\rangle_B$$

Cioè sia U un gate in grado di copiare lo stato del qubit A nel qubit B. Allora si avrà anche che:

$$U|00\rangle = |00\rangle$$

e

$$U|10\rangle = |11\rangle$$

Per linearità quindi avremo:

$$U(\frac{|00\rangle + |10\rangle}{\sqrt{2}}) = \frac{U|00\rangle + U|10\rangle}{\sqrt{2}} = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

Ma:

$$\frac{|00\rangle + |10\rangle}{\sqrt{2}} = \frac{|0\rangle_A + |1\rangle_A}{\sqrt{2}} \otimes |0\rangle_B$$

E quindi dovremmo avere anche:

$$U(\frac{|00\rangle + |10\rangle}{\sqrt{2}}) = U(\frac{|0\rangle_A + |1\rangle_A}{\sqrt{2}} \otimes |0\rangle_B) = \frac{|0\rangle_A + |1\rangle_A}{\sqrt{2}} \otimes \frac{|0\rangle_B + |1\rangle_B}{\sqrt{2}} \neq \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

□

Al contrario dei computer classici, in cui è possibile copiare lo stato di un bit, nei computer quantistici non è possibile copiare lo stato di un qubit qualsiasi (a meno di conoscere a priori lo stato del qubit da copiare).
Quindi Eve può soltanto applicare gate ed effettuare misure sui qubit di Alice e Bob. Qual è la miglior strategia che può adottare? Vedremo di seguito che ogni tentativo comporterà una variazione del valore S ben sotto la soglia classica di 2.

3.2.1 Strategia 1: Misura di tutti i qubit di Bob

La strategia più ingenua per Eve è misurare semplicemente tutti i qubit di Bob. Questo ci permette di vedere facilmente la differenza dei valori di S se qualcuno cerca di rompere l'entanglement togliendo i qubit da uno stato di sovrapposizione.
Tradotta in codice, questa strategia viene applicata nel seguente modo, prima dell'applicazione dei gate da parte di Alice e Bob:

```
def eveAction1(circuits: list):
    # Strategia 1: Misuro tutti i qubit di Bob
    for circuit in circuits:
        circuit.measure(1, 1)
```

Eseguendo ancora 10 prove, con $n = 500$, il simulatore restituisce i risultati riportati in Tabella 3.6:

S	Δ	L	Errori chiave
1.22	-1.61	113	19
1.58	-1.24	117	14
1.35	-1.48	112	19
1.41	-1.41	125	14
0.95	-1.87	100	9
1.43	-1.40	108	14
1.60	-1.23	101	10
1.36	-1.47	104	12
1.34	-1.49	129	16
1.68	-1.15	131	24

Tabella 3.6: Dati raccolti col simulatore con 500 coppie di qubit, strategia 1 di Eve

In questa tabella vengono mostrati anche il numero di bit errati nella chiave ottenuta da Bob rispetto alla chiave di Alice. Nei precedenti dati delle misure (quelli ottenuti senza l'intervento di Eve) questo risultato non è stato incluso nelle varie tabelle perché era sempre 0: il simulatore è perfetto ed è privo di rumore, dunque non potrà mai esserci una discrepanza nella chiave condivisa senza l'intervento di Eve.
Si osserva che il valore di S in questo caso è sempre minore di 2: nel caso "peggiore" vale 1,68.
Si è scelto $n = 500$ perché nel caso senza Eve era il valore di n minore che assicurava una buona precisione di S.

Per completezza, vengono inclusi nella Tabella 3.7 i risultati con $n = 100$:

S	Δ	L	Errori chiave
0.59	-2.24	18	4
0.97	-1.86	21	4
0.90	-1.92	29	2
1.66	-1.17	23	5
1.70	-1.13	29	4
0.27	-2.56	22	2
1.62	-1.21	33	2
0.73	-2.09	19	1
1.51	-1.32	15	2
0.51	-2.32	25	3

Tabella 3.7: Dati raccolti col simulatore con 100 coppie di qubit, strategia 1 di Eve

Di seguito, in Tabella 3.8, invece gli stessi risultati ottenuti però con $n = 2000$, cioè con la precisione massima tra i valori di n che erano stati scelti in precedenza:

S	Δ	L	Errori chiave
1.47	-1.35	445	45
1.33	-1.50	432	51
1.55	-1.28	402	49
1.37	-1.46	459	65
1.40	-1.43	466	58
1.57	-1.25	435	57
1.41	-1.41	433	58
1.53	-1.30	477	59
1.47	-1.36	475	69
1.57	-1.26	428	55

Tabella 3.8: Dati raccolti col simulatore con 2000 coppie di qubit, strategia 1 di Eve

Questa volta S non supera il valore di 1.57. Di seguito, in Tabella 3.9, il riassunto della media delle diverse misure:

n	μ	σ	$\lvert\Delta_\mu\rvert$	$\lvert\Delta_\mu\rvert - \sigma$
100	1.04	0.51	1.78	1.28
500	1.39	0.20	1.44	1.24
2000	1.47	0.08	1.36	1.28

Tabella 3.9: Medie dei valori S sul simulatore con la strategia 1 di Eve

Dove l'ultima colonna vogliamo che sia sempre maggiore di $2\sqrt{2}-2 = 0.83$

3.2.2 Strategia 2: Applicazione di alcuni gate prima della misura dei qubit

Qualsiasi combinazione di gate e misure rovina la correlazione misurata dal valore S.
Non esiste nessuna strategia efficace in questo senso; qui di seguito un altro esempio di strategia in cui Eve cerca di replicare il comportamento di Bob applicando casualmente gli stessi gate che applicherebbe lui:

```
def eveAction2(circuits: list):
    # Strategia 2: Misuro tutti i qubit di Bob usando una direzione
    ↪  casuale tra quelle a disposizione di Bob
    for circuit in circuits:
        eve_basis = random.choice([1, 2, 3])
        if eve_basis == 1:
            B1(circuit)
        elif eve_basis == 2:
            B2(circuit)
        elif eve_basis == 3:
            B3(circuit)
        circuit.measure(1, 1)
```

S	Δ	L	Errori chiave
1.51	-1.32	119	26
1.20	-1.63	96	17
0.93	-1.90	121	30
1.07	-1.76	113	19
0.95	-1.88	115	21
1.30	-1.52	103	18
1.32	-1.51	110	32
1.07	-1.75	128	23
1.56	-1.27	112	25
1.04	-1.78	106	22

Tabella 3.10: Dati raccolti col simulatore con 500 coppie di qubit, strategia 2 di Eve

Il risultato in questo caso è ancora peggiore (per Eve): la media di S per i risultati mostrati in Tabella 3.10 è inferiore rispetto alla strategia 1 (1.20 vs 1.39).

È inutile soffermarsi su strategie di questo tipo, è più interessante invece il caso in cui anche Eve tenti di utilizzare le proprietà dell'entanglement.

3.2.3 Strategia 3: Entanglement di un terzo qubit di Eve

Sapendo che lo stato iniziale dei qubit inviati da Charlie è:

$$|\psi\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}}$$

Eve potrebbe mettere in entanglement un proprio qubit, e ottenere uno stato del genere:

$$|\psi'\rangle = \frac{|011\rangle - |100\rangle}{\sqrt{2}}$$

Inviando poi i primi due qubit ad Alice e Bob, potrebbe aspettare che questi ultimi effettuino le loro misure, condividano le basi usate per la misura, e ottenere informazioni sulla chiave privata condivisa.
Per ottenere lo stato $|\psi'\rangle$ è sufficiente applicare una porta $CNOT$ che abbia come control-qubit il qubit di Bob e come qubit bersaglio quello di Eve; il codice è il seguente:

```
def eveAction3(circuits: list):
    # Strategia 3: Metto in entanglement un terzo qubit posseduto da
    ↪  Eve con il qubit di Bob
    for circuit in circuits:
        circuit.cnot(1, 2)
```

Per esempio, uno dei 9 circuiti generati con questa strategia è il seguente:

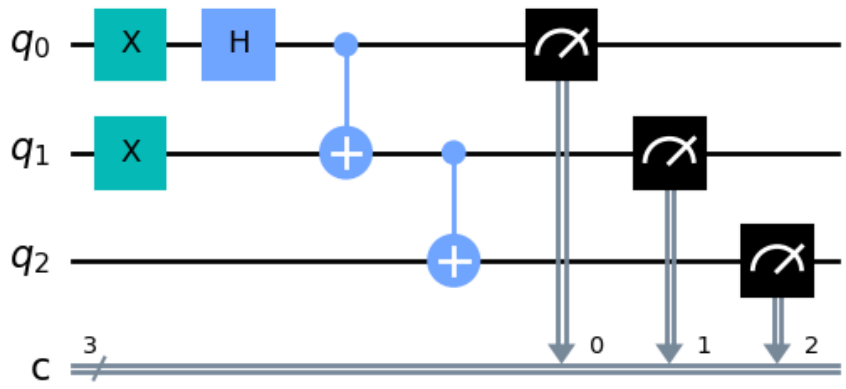

Figura 3.2: Circuito con A_3 e B_2 applicati da Alice e Bob, e la terza strategia di Eve

Questo metodo è in effetti efficace se la base di misura fosse sempre nella direzione Z ($\phi = \pi/2$). In questo caso lo stato su cui Alice e Bob effettuano le misure è ancora $|\psi\rangle$, e Eve sa che misurando il suo qubit lungo l'asse Z

otterrà lo stesso identico risultato di Bob.
Ma Alice e Bob non utilizzano soltanto l'asse Z per ottenere la chiave, utilizzano anche l'asse $(Z + X)/\sqrt{2}$ ($\phi = \pi/4$).
In questi casi Alice e Bob applicano diversi gate prima di effettuare la misura, tra cui il gate Hadamard (H), e l'entanglement di Eve rovina l'entanglement di Alice e Bob, che otterranno dunque dei risultati incoerenti.
Ricordando l'effetto del gate H:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \equiv |+\rangle$$

$$H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \equiv |-\rangle$$

Si noti com'è sufficiente l'applicazione di H da parte di Alice e Bob sullo stato $|\psi'\rangle$ per rovinare l'entanglement: (ignoriamo i coefficienti di normalizzazione per chiarezza di esposizione)

$$\begin{aligned} H_A H_B |\psi'\rangle &= H_A \Big[|0\rangle_A \otimes \big(|0\rangle - |1\rangle\big)_B \otimes |1\rangle_E - |1\rangle \otimes \big(|0\rangle + |1\rangle\big)_B \otimes |0\rangle_E \Big] \\ &= H_A \Big[|001\rangle - |011\rangle - |100\rangle - |110\rangle \Big] \\ &= |001\rangle + |101\rangle - |011\rangle - |111\rangle - |000\rangle + |100\rangle - |010\rangle + |110\rangle \end{aligned} \quad (3.1)$$

Ora Alice e Bob non otterranno più dei risultati correlati: uno 0 di Alice può corrispondere sia a uno 0 che a un 1 di Bob, e viceversa.

Senza il terzo qubit avremmo invece:

$$H_A H_B |\psi\rangle = |00\rangle + |10\rangle - |01\rangle - |11\rangle - |00\rangle + |10\rangle - |01\rangle \propto |10\rangle - |01\rangle \quad (3.2)$$

Cioè i risultati di Alice e Bob saranno ancora anti-correlati.

Inoltre, non vengono rovinati solamente i qubit nella direzione $\pi/4$, usati come chiave, ma anche tutti gli altri qubit utilizzati per il calcolo del valore S, che cambierà e tornerà a essere minore di 2 come riportato qui di seguito dai dati raccolti in Tabella 3.11 e in Tabella 3.12:

S	Δ	L	Errori chiave
1.89	-0.94	119	15
1.16	-1.67	109	17
1.25	-1.58	104	15
1.23	-1.60	107	11
1.59	-1.24	112	22
1.24	-1.58	112	11
1.16	-1.67	103	9
1.31	-1.52	116	15
1.16	-1.67	113	13
1.08	-1.75	92	12

Tabella 3.11: Dati raccolti col simulatore con 500 coppie di qubit, strategia 3 di Eve

n	μ	σ	$\lvert\Delta_\mu\rvert$	$\lvert\Delta_\mu\rvert - \sigma$
500	1.31	0.24	1.52	1.29

Tabella 3.12: Media dei valori S sul simulatore con la strategia 3 di Eve

Anche in questo caso, il tentativo di Eve non le permette né di ottenere la chiave corretta né di passare inosservata, con un valore di S ben sotto la soglia di 2.

Di seguito, in Figura 3.3, un confronto tra le misure senza Eve e le misure con le 3 strategie di Eve:

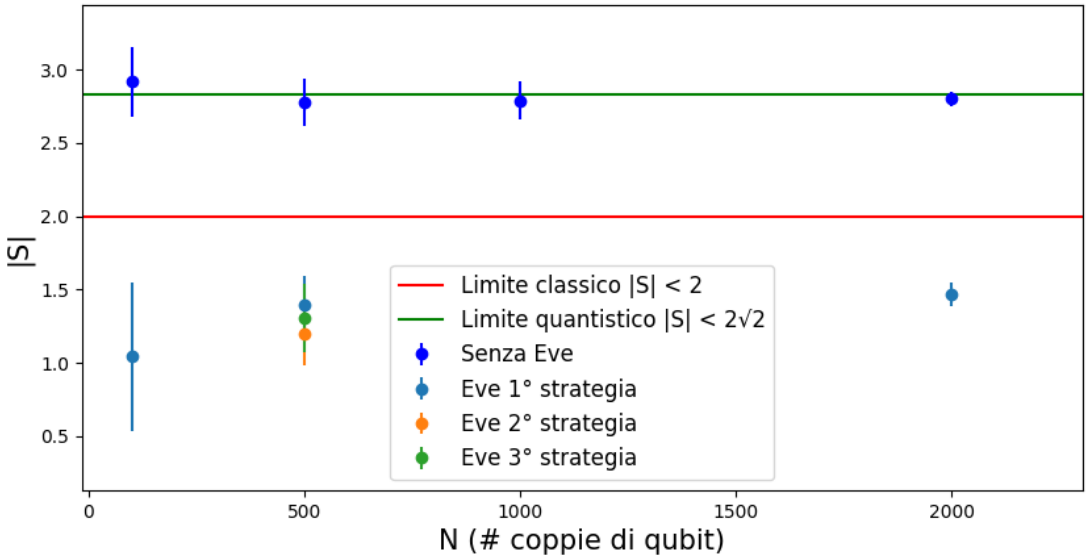

Figura 3.3: Confronto valori medi di S con e senza l'intervento di Eve, sul simulatore

Capitolo 4

Esecuzione del protocollo E91 su computer quantistici reali di IBM

4.1 Backend di IBM e loro caratteristiche

IBM mette a disposizione diversi computer reali (backend) i quali possono eseguire i circuiti quantistici generati con Qiskit.
Ogni backend ha il nome di una città: "ibm_lagos" e "ibmq_jakarta" sono stati scelti per l'esecuzione di questo protocollo.
Per ogni backend, IBM specifica alcune caratteristiche. Le più importanti sono il QV (Quantum Volume), l'errore sui $CNOT$, l'errore sulle letture, il tempo medio $T1$ e il tempo medio $T2$, il CLOPS e il numero di qubit disponibili in totale.
Per entrambi i backend, il numero di qubit è 7; il nostro programma ne utilizza solo 3, quindi questo parametro non crea nessun problema.
Per il Quantum Volume esistono diverse definizioni, ma quella attualmente utilizzata come standard è la definizione di IBM:

$$\log_2 V_Q = \arg\max_{n \leq N} \{\min [n, d(n)]\}$$

Dove N è il numero massimo di qubit; d è la profondità di un circuito, cioè il numero massimo di step eseguibili dal computer utilizzando n qubit con una precisione accettabile.
Il Quantum Volume dà quindi un'indicazione di quanto è precisa la computazione di un circuito con un alto numero di qubit coinvolti.

L'errore dei $CNOT$ è l'errore medio che si ha nell'applicare un gate $CNOT$ su due qubit: c'è quindi una certa probabilità che il gate non inverta il qubit bersaglio come dovrebbe.

L'errore sulle letture è la probabilità media di ricevere come output un bit opposto rispetto a quello che ci si aspetta.

Il tempo $T1$ è il tempo medio di decoerenza "longitudinale", cioè la costante di tempo di decadimento del qubit nel suo ground-state. La probabilità di rimanere nello stato eccitato è:

$$P_{T1}(t) = e^{-t/T1}$$

Il tempo $T2$ è il tempo medio di decoerenza di fase. In maniera simile al tempo $T1$, il tempo $T2$ misura il tempo medio in cui un qubit in stato di sovrapposizione rimane con la stessa fase. Indica quindi la capacità di un computer di mantenere uno stato $|+\rangle$ prima che diventi $|-\rangle$, o viceversa.

Il CLOPS, ovvero Circuit Layer Operations Per Second, misura quanti circuiti al secondo possono essere eseguiti dal computer. Questo parametro, insieme ai $T1$ e $T2$, influiscono sulla precisione dei risultati di un circuito.

In generale, il QV è un buon indicatore che riassume un po' tutti i parametri: a QV più alto corrispondono prestazioni migliori.

I computer di IBM vengono calibrati più volte per ottenere i parametri sopra citati. Di seguito in Tabella 4.1 un confronto tra le caratteristiche di ibmq_jakarta e ibm_lagos:

Parametro	ibmq_jakarta	ibm_lagos
QV	16	32
T1	162.23 μs	137.84 μs
T2	37.12 μs	86.87 μs
CLOPS	2400	2700
Median readout err.	2.430e-2	1.540e-2
Median CNOT err.	7.276e-3	6.693e-3

Tabella 4.1: Confronto delle caratteristiche tra ibmq_jakarta e ibm_lagos

Di seguito, nelle Figure 4.1 e 4.2, viene mostrata la disposizione topologica dei qubit per i due computer e i loro errori di misura (readout assignment) e di applicazione dei gate $CNOT$:

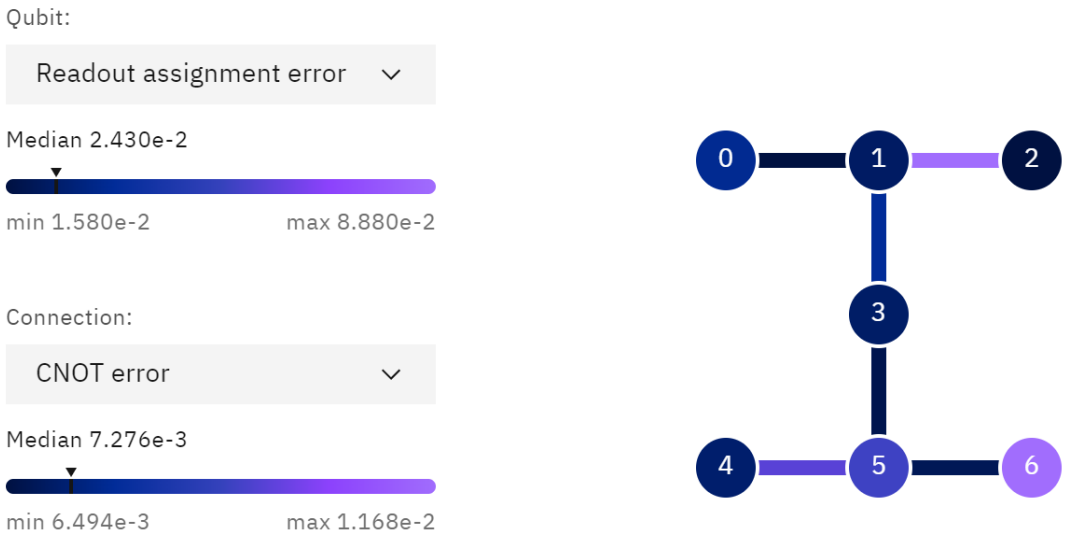

Figura 4.1: Disposizione dei qubit per ibmq_jakarta

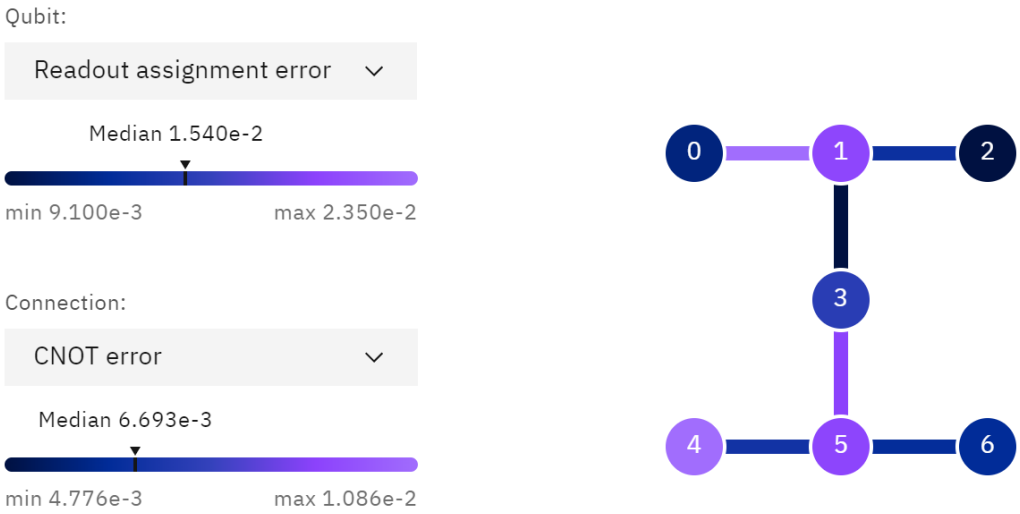

Figura 4.2: Disposizione dei qubit per ibm_lagos

4.2 Necessità di un programma ottimizzato

I computer di IBM pongono un limite al numero di circuiti eseguibili durante uno stesso job. Per ibmq_jakarta questo limite è 300, il che rende impossibile prendere la maggior parte delle misure che sono state acquisite sul simulatore. Per questo è stato scritto un secondo programma ottimizzato, il quale anziché eseguire N circuiti ne esegue solamente 9 con N conteggi, distribuiti casualmente come in precedenza.
Il funzionamento di questo secondo programma è identico a quello di prima, con l'unico difetto che non riesce a riprodurre la strategia numero 2 di Eve, che richiederebbe più di 9 circuiti diversi. Le strategie 1 e 3 invece sono ancora replicabili e sono sufficienti per analizzare il comportamento del protocollo.

Il codice del programma ottimizzato è presente nell'appendice B.

4.3 Risultati delle misure sui computer di IBM

4.3.1 Risultati senza Eve su ibmq_jakarta

Di seguito vengono riportati i risultati ottenuti su ibmq_jakarta con le stesse configurazioni usate sul simulatore. Si noti che nelle Tabelle 4.2, 4.3, 4.4 e 4.5 gli errori nella chiave condivisa compaiono anche senza l'intervento di Eve, al contrario di quanto succedeva sul simulatore.

S	Δ	L	Errori chiave
2.19	-0.64	20	2
2.84	0.01	14	1
2.00	-0.83	16	1
3.42	0.59	25	1
2.54	-0.29	17	0
3.01	0.18	27	0
2.92	0.09	18	0
1.96	-0.87	21	0
2.71	-0.11	22	0
3.15	0.32	17	1

Tabella 4.2: Dati raccolti su ibmq_jakarta con $n = 100$, senza Eve

Nella Tabella 4.2 vengono visualizzati i risultati per $n = 100$, dove 2 volte su 10 sono stati ottenuti dei valori di $|S| \leq 2$ (2.00 e 1.96).

S	Δ	L	Errori chiave
2.58	-0.25	104	5
2.39	-0.43	114	4
2.59	-0.24	111	7
2.41	-0.42	127	14
2.58	-0.25	123	10
2.61	-0.22	100	6
2.55	-0.28	111	14
2.55	-0.28	88	7
2.61	-0.22	95	4
2.47	-0.36	119	6

Tabella 4.3: Dati raccolti su ibmq_jakarta con $n = 500$, senza Eve

Nella Tabella 4.3, con $n = 500$, tutti i valori di $|S|$ risultano maggiori di 2. Questo permette di sapere che Eve non è intervenuta.

S	Δ	L	Errori chiave
2.44	-0.39	209	10
2.47	-0.36	213	12
2.67	-0.16	223	12
2.62	-0.21	219	15
2.38	-0.45	204	10
2.46	-0.37	227	10
2.43	-0.40	230	11
2.65	-0.18	212	16
2.56	-0.27	209	10
2.41	-0.42	247	14

Tabella 4.4: Dati raccolti su ibmq_jakarta con $n = 1000$, senza Eve

S	Δ	L	Errori chiave
2.49	-0.34	473	22
2.43	-0.40	428	25
2.64	-0.19	446	32
2.59	-0.24	452	20
2.60	-0.23	454	18
2.25	-0.58	414	22
2.53	-0.30	458	20
2.52	-0.31	481	25
2.22	-0.61	473	16
2.55	-0.28	471	24

Tabella 4.5: Dati raccolti su ibmq_jakarta con $n = 2000$, senza Eve

Nelle Tabelle 4.4 e 4.5, che si riferiscono ai dati raccolti per $n = 1000$ e $n = 2000$, rispettivamente, si osserva che ogni valore di $|S|$ risulta maggiore di 2, come già constatato per $n = 500$.

4.3.2 Risultati senza Eve su ibm_lagos

Di seguito gli stessi dati ma ottenuti da ibm_lagos:

S	Δ	L	Errori chiave
2.77	-0.06	22	0
2.54	-0.29	18	1
2.59	-0.24	27	0
1.83	-1.00	22	1
2.67	-0.16	15	0
2.68	-0.15	25	0
2.34	-0.49	20	0
2.77	-0.06	25	0
1.72	-1.11	26	0
2.40	-0.43	25	0

Tabella 4.6: Dati raccolti su ibm_lagos con $n = 100$, senza Eve

Anche su ibm_lagos, per $n = 100$, la Tabella 4.6 evidenzia come 2 risultati su 10 producano un valore di $|S| \leq 2$ (1.83 e 1.72).

S	Δ	L	Errori chiave
2.60	-0.22	104	5
2.73	-0.10	105	1
2.79	-0.03	93	4
2.62	-0.21	105	0
2.70	-0.12	120	4
2.78	-0.05	126	1
2.51	-0.32	106	2
2.65	-0.17	97	6
2.90	0.08	109	2
2.54	-0.29	107	6

Tabella 4.7: Dati raccolti su ibm_lagos con $n = 500$, senza Eve

Nella Tabella 4.7 si può osservare che, per $n = 500$, anche su ibm_lagos si ottengono dei valori di $|S|$ sempre maggiori di 2.

Nelle Tabelle 4.8 e 4.9 qui di seguito vengono riportati i dati per $n = 1000$ e $n = 2000$. Di nuovo, non troviamo più nessun valore di $|S| \leq 2$:

S	Δ	L	Errori chiave
2.34	-0.49	195	9
2.63	-0.20	237	9
2.73	-0.09	254	9
2.74	-0.09	250	10
2.69	-0.14	213	11
2.70	-0.13	187	6
2.70	-0.13	204	7
2.77	-0.06	224	5
2.53	-0.30	231	5
2.58	-0.25	213	4

Tabella 4.8: Dati raccolti su ibm_lagos con $n = 1000$, senza Eve

S	Δ	L	Errori chiave
2.83	0.00	432	11
2.77	-0.06	416	17
2.53	-0.30	467	18
2.64	-0.18	421	10
2.76	-0.06	463	19
2.77	-0.06	466	27
2.69	-0.14	397	17
2.72	-0.11	462	19
2.65	-0.18	430	16
2.76	-0.07	458	17

Tabella 4.9: Dati raccolti su ibm_lagos con $n = 2000$, senza Eve

4.3.3 Risultati con l'intervento di Eve su ibmq_jakarta

Vengono riportati di seguito i risultati delle strategie 1 e 3 di Eve su ibmq_jakarta:

S	Δ	L	Errori chiave
1.94	-0.89	29	7
1.83	-1.00	29	10
1.37	-1.46	28	6
1.44	-1.38	16	4
0.40	-2.43	19	4
1.86	-0.97	22	2
1.32	-1.51	25	4
0.89	-1.94	14	3
0.39	-2.44	29	7
1.65	-1.18	17	4

Tabella 4.10: Dati raccolti su ibmq_jakarta con $n = 100$, strategia 1 di Eve

Nella Tabella 4.10 vengono riportati i risultati per $n = 100$ utilizzando la strategia 1 di Eve. I valori di $|S|$ sono tutti minori di 2, ma alcuni valori si avvicinano più di altri alla soglia (1.94 è inferiore di soli 6 centesimi alla soglia).

Nelle Tabelle 4.11 e 4.12 gli stessi dati sono stati presi con $n = 500$ e $n = 2000$, rispettivamente. Di nuovo, tutti i valori di $|S|$ sono minori di 2, e il valore più alto che si trova è 1.69 per $n = 500$:

S	Δ	L	Errori chiave
1.09	-1.74	102	17
1.19	-1.63	131	24
1.15	-1.68	118	27
1.69	-1.14	111	26
1.12	-1.71	104	21
1.25	-1.57	117	22
1.53	-1.29	120	22
1.39	-1.44	111	18
1.49	-1.34	117	20
0.95	-1.88	137	18

Tabella 4.11: Dati raccolti su ibmq_jakarta con $n = 500$, strategia 1 di Eve

S	Δ	L	Errori chiave
1.19	-1.64	407	60
1.36	-1.47	444	81
1.18	-1.65	420	85
1.00	-1.82	394	85
1.28	-1.55	440	88
1.14	-1.68	437	76
1.02	-1.80	408	70
1.22	-1.60	428	73
0.96	-1.87	460	73
1.12	-1.71	452	86

Tabella 4.12: Dati raccolti su ibmq_jakarta con $n = 2000$, strategia 1 di Eve

Per la strategia 3 sono stati raccolti dati solamente per $n = 500$, visualizzati nella Tabella 4.13:

S	Δ	L	Errori chiave
0.85	-1.97	116	24
1.12	-1.71	115	14
1.09	-1.73	104	11
1.34	-1.49	112	25
1.05	-1.77	110	19
0.81	-2.02	100	25
0.92	-1.91	95	16
1.37	-1.46	111	18
1.03	-1.80	128	16
1.22	-1.60	117	16

Tabella 4.13: Dati raccolti su ibmq_jakarta con $n = 500$, strategia 3 di Eve

Nessun valore di $|S|$ raggiunge la soglia di 2.

4.3.4 Risultati con l'intervento di Eve su ibm_lagos

Di seguito i risultati delle strategie 1 e 3 di Eve su ibm_lagos:

S	Δ	L	Errori chiave
2.00	-0.83	21	4
1.45	-1.38	21	4
1.96	-0.86	22	4
1.37	-1.46	29	1
1.30	-1.53	20	2
1.34	-1.49	21	2
2.95	0.12	26	7
1.71	-1.12	23	1
1.60	-1.23	27	3
0.44	-2.39	24	8

Tabella 4.14: Dati raccolti su ibm_lagos con $n = 100$, strategia 1 di Eve

Nella Tabella 4.14 vengono mostrati i dati raccolti con la strategia 1 di Eve per $n = 100$ su ibm_lagos. Come per ibmq_jakarta, anche su ibm_lagos ci sono dei problemi nel rivelare la presenza di Eve con $n = 100$: due valori di $|S|$ risultano maggiori o uguale a 2 (2.00 e 2.95).

Per $n = 500$ e $n = 2000$ invece il problema non sussiste, come evidenziato dalle Tabelle 4.11 e 4.12:

S	Δ	L	Errori chiave
1.36	-1.47	103	19
1.35	-1.47	114	12
1.16	-1.67	132	31
1.27	-1.56	120	15
1.67	-1.16	115	14
1.22	-1.61	108	12
1.11	-1.72	105	25
1.47	-1.35	110	20
0.83	-2.00	114	16
1.39	-1.44	102	14

Tabella 4.15: Dati raccolti su ibm_lagos con $n = 500$, strategia 1 di Eve

S	Δ	L	Errori chiave
1.18	-1.65	440	70
1.25	-1.58	446	72
1.32	-1.51	455	68
1.00	-1.83	458	70
1.10	-1.73	479	74
1.28	-1.55	449	77
1.18	-1.65	416	76
1.69	-1.14	444	67
1.24	-1.59	469	79
1.31	-1.51	430	76

Tabella 4.16: Dati raccolti su ibm_lagos con $n = 2000$, strategia 1 di Eve

Nella Tabella 4.17, infine, i risultati di ibm_lagos per $n = 500$ con la strategia 3 di Eve:

S	Δ	L	Errori chiave
1.47	-1.35	108	20
1.31	-1.52	121	22
1.09	-1.74	113	18
1.19	-1.64	114	22
1.43	-1.40	103	14
0.95	-1.87	104	12
1.30	-1.53	115	19
1.15	-1.68	117	17
1.22	-1.61	112	16
1.43	-1.40	125	17

Tabella 4.17: Dati raccolti su ibm_lagos con $n = 500$, strategia 3 di Eve

Anche in questo caso, nessun valore di $|S|$ raggiunge la soglia di 2.

4.4 Confronto tra ibm_lagos e ibmq_jakarta

Come previsto dai parametri indicati da IBM, ibm_lagos è risultato essere più preciso di ibmq_jakarta. Nella Tabella 4.18 viene visualizzato un confronto tra i valori medi di $|S|$ dei due computer:

n	μ_S (Jakarta)	μ_S (Lagos)
100	2.67 ± 0.47	2.43 ± 0.36
500	2.53 ± 0.08	2.68 ± 0.12
1000	2.51 ± 0.10	2.64 ± 0.13
2000	2.48 ± 0.14	2.71 ± 0.08

Tabella 4.18: Medie dei valori S sui computer reali (no Eve)

Escludendo i risultati per $n = 100$, dove i qubit sono troppo pochi per avere una statistica affidabile, i valori di $|S|$ di ibm_lagos sono sempre più vicini a $2\sqrt{2}$ rispetto a quelli di ibmq_jakarta.
Si noti anche che alcune esecuzioni singole del protocollo con $n = 100$ hanno restituito un valore di $|S|$ minore o uguale a 2, sia per ibmq_jakarta che per ibm_lagos. Questo significa che non è possibile distinguere l'assenza o la presenza di Eve con un numero di qubit così basso. Da $n = 500$ i risultati sono già migliori e $|S|$ non arriva mai al di sotto di 2.
Nonostante ciò, la media di $|S|$ non è mai compatibile con $2\sqrt{2}$ ma risulta sempre inferiore, come messo in evidenza dalla Figura 4.3:

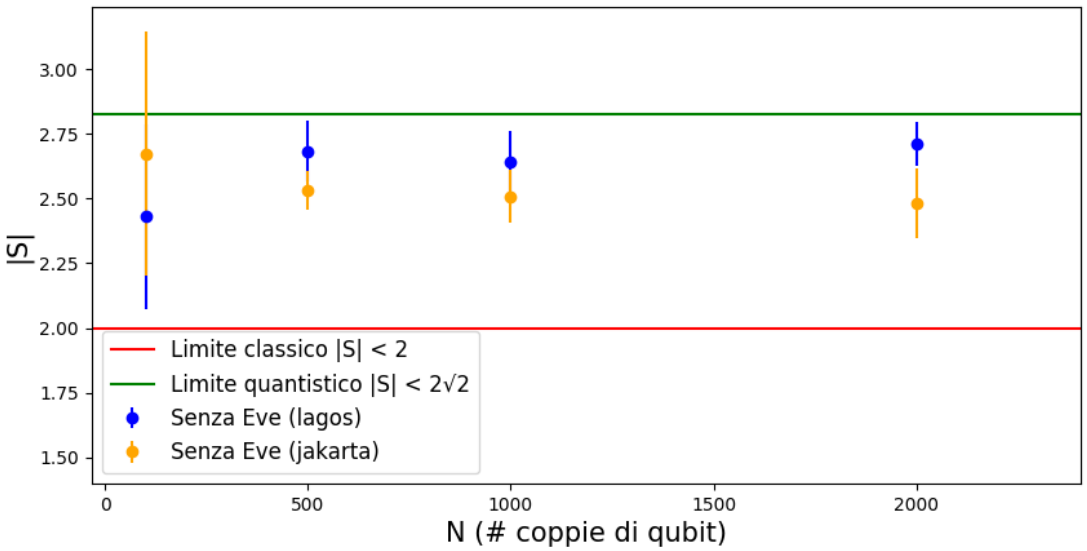

Figura 4.3: Confronto valori medi di S senza l'intervento di Eve, su ibm_lagos e ibmq_jakarta

Dal punto di vista pratico, questo non è un problema: infatti è sufficiente che $|S|$ sia maggiore di 2 per escludere la presenza di Eve. Vediamo dai dati nella Tabella 4.19 che con una qualunque strategia di Eve il valore di $|S|$ è sempre minore di 2. Di nuovo c'è qualche problema con $n = 100$, e

qualche volta capitano valori di $|S|$ maggiori di 2 anche con l'intervento di Eve, confermando il fatto che n deve valere almeno 500 per avere dei risultati attendibili.

n	# Strategia	μ_S (Jakarta)	μ_S (Lagos)
100	1	1.31 ± 0.54	1.61 ± 0.61
500	1	1.29 ± 0.22	1.28 ± 0.22
2000	1	1.15 ± 0.12	1.25 ± 0.17
500	3	1.08 ± 0.18	1.25 ± 0.16

Tabella 4.19: Medie dei valori S sui computer reali con l'intervento di Eve

La Figura 4.4 mostra un riassunto di tutti i dati presi da ibm_lagos e ibmq_jakarta, con e senza l'intervento di Eve:

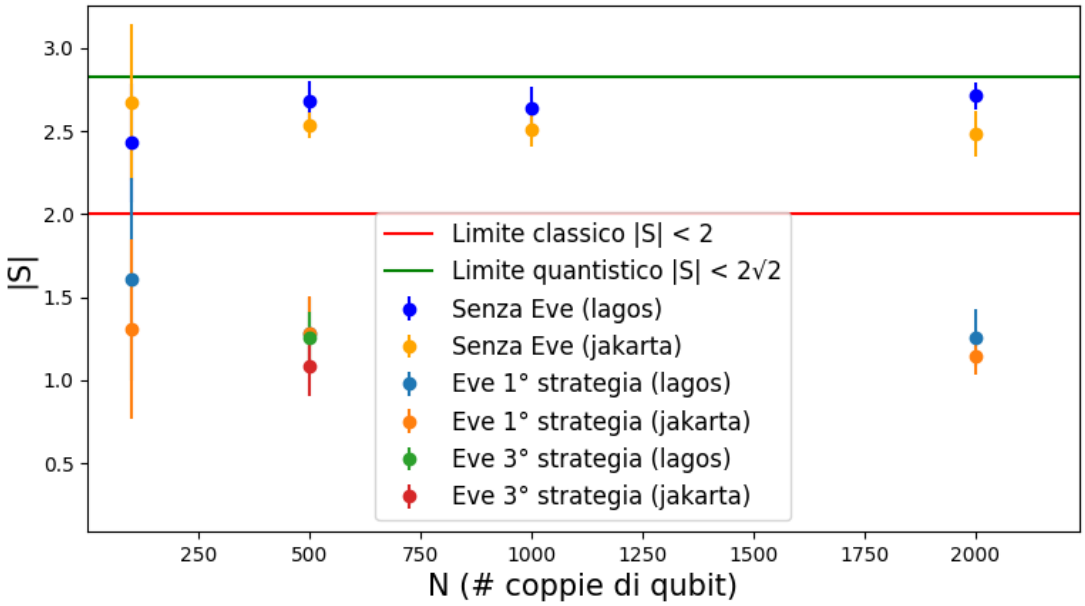

Figura 4.4: Confronto valori medi di S con e senza l'intervento di Eve, su ibm_lagos e ibmq_jakarta

Lo scambio di chiavi è quindi sicuro, e l'intervento di Eve può anche essere scoperto calcolando S se il numero di qubit è sufficiente. La criticità adesso è un'altra: l'imprecisione della chiave stessa.
In assenza di Eve, Alice e Bob devono poter scambiare la chiave privata senza problemi, ma notiamo dai dati che solo per $n = 500$, solo in un caso su 10, c'è stato uno scambio di chiave perfetto, senza discrepanze tra i bit di Alice e di Bob. Aumentando il numero di qubit aumenta anche il numero di bit in errore, in quanto l'errore introdotto dalla misura dei qubit di un computer quantistico reale ha una certa probabilità ed è quindi proporzionale alla quantità di misure che si effettuano.
Può essere quindi utile guardare la percentuale media di errori al variare di n:

n	$\mu_{err\%}$ (Jakarta)	$\mu_{err\%}$ (Lagos)
100	3.33 ± 3.61	1.01 ± 2.03
500	6.96 ± 2.83	2.97 ± 2.07
1000	5.47 ± 0.96	3.40 ± 1.02
2000	4.94 ± 1.01	3.86 ± 0.90

Tabella 4.20: Medie degli errori nella chiave sui computer reali

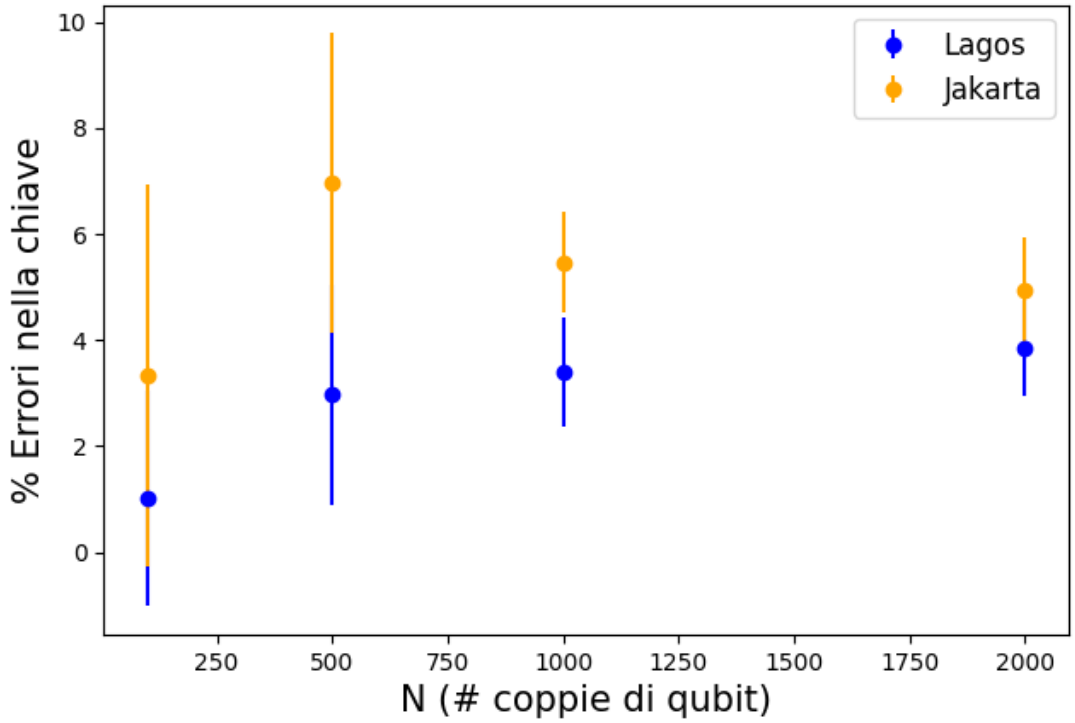

Figura 4.5: Confronto % errore medio nella chiave condivisa senza l'intervento di Eve, su ibm_lagos e ibmq_jakarta

Di nuovo, ibm_lagos è il più preciso, ma con un errore che non scende sotto al 3%.
Siamo quindi incontro a un limite tecnologico: l'imprecisione di lettura e la decoerenza dello stato di un qubit.
Esistono circuiti che implementano tecniche di correzione degli errori, ma impiegano l'utilizzo di diversi qubit fisici per un singolo qubit logico e qui incontriamo un altro limite tecnologico: ibm_lagos e ibmq_jakarta hanno soltanto 7 qubit. È necessario quindi l'utilizzo di computer con caratteristiche migliori per ottenere risultati più precisi.
Alice e Bob non possono neanche sapere di aver letto una chiave errata, perché per saperlo dovrebbero comunicare pubblicamente l'intera chiave privata ottenuta, che invece deve rimanere segreta.

4.5 Confronto con BB84

Il protocollo più famoso di QKD è il BB84. Nato nel 1984 da Bennett e Brassard, ottiene lo stesso risultato dell'E91: lo scambio sicuro di una chiave privata con la possibilità di identificare un'ipotetica Eve.[1]
Gli step del protocollo BB84 sono i seguenti:

1. Alice decide di inviare dei dati a Bob e genera quindi una serie di N bit casuali;
2. Alice sceglie poi una base per ogni bit, a caso tra Z e X;
3. Per ogni base Z, il bit corrispondente viene codificato come un qubit nello stato $|0\rangle$ o $|1\rangle$ per i valori 0 e 1, rispettivamente. Per ogni base X, il bit corrispondente viene codificato in un qubit nello stato $|+\rangle$ o $|-\rangle$ per i valori 0 e 1, rispettivamente;
4. Alice invia i qubit a Bob. In questa comunicazione potrebbe esserci l'intervento di Eve;
5. Bob misura i qubit scegliendo una base a caso tra Z e X;
6. Alice e Bob comunicano le basi che hanno scelto. I bit ottenuti utilizzando la stessa base avranno anche lo stesso valore, e saranno utilizzabili per la chiave privata;
7. Per scoprire l'intervento di Eve, Alice e Bob comunicano anche metà della stessa chiave privata; se corrisponde per intero, non c'è stato l'intervento di Eve e il resto della chiave, quello che non è stato comunicato, può essere utilizzato per cifrare un messaggio;
8. Se ci sono degli errori nella parte di chiave comunicata pubblicamente, è possibile che Eve abbia tentato di interferire. Alice e Bob devono quindi cercare un diverso canale di comunicazione più sicuro.

Il meccanismo per occultare la chiave a Eve è il medesimo: l'utilizzo di due basi diverse per la chiave privata impedisce a Eve di sapere in anticipo come misurare i qubit.
Il risultato è identico: la chiave rimane sicura e Eve non ha modo né di accedere alla chiave né di interferire senza essere scoperta.

C'è da notare però che il BB84 utilizza un numero inferiore di qubit per ottenere la stessa lunghezza di chiave, rispetto al protocollo E91. Nel BB84 in media la metà dei qubit viene acquisita con la stessa base del mittente, e la metà di questi qubit viene infine tenuta segreta per essere utilizzata come chiave. Quindi avremo una lungehzza media della chiave:

$$\langle L \rangle = \frac{1}{4} N = 0.25N$$

E quindi per ottenere una chiave lunga L si useranno in media $\langle N \rangle$ qubit:

$$\langle N \rangle = 4L$$

Mentre per l'E91 abbiamo:

$$\langle L \rangle = \frac{2}{9} N = 0.22N$$

$$\langle N \rangle = 4.5L$$

Perché le combinazioni di basi diverse sono in tutto 9 e solo in 2 casi su 9 Alice e Bob scelgono la stessa base.
A essere precisi, per ogni bit della chiave vengono generati due qubit in entanglement, quindi il totale di qubit generati e inviati da Charlie sarà più del doppio dei qubit generati e inviati da Alice nel BB84!

$$\langle N \rangle = 9L$$

Il vantaggio dell'E91 però è che le persone che comunicano, Alice e Bob, non devono necessariamente possedere un dispositivo in grado di generare e inviare qubit; è sufficiente che posseggano un dispositivo in grado di ricevere e misurare qubit su 3 basi diverse da un ente terzo (Charlie).
Questo vantaggio permetterebbe una diffusione del protocollo E91 anticipata rispetto al BB84, avendo dei requisiti minimi tecnologici inferiori.

Conclusione

Fino ad oggi, l'utilizzo di cifrature a chiave asimmetrica si basava sulla difficoltà di computazione di alcuni problemi matematici. Con il passare del tempo anche la potenza di calcolo dei computer classici cresce; i protocolli di comunicazione devono quindi essere modificati in modo che la chiave privata sia più difficile da calcolare.
L'avvento dei computer quantistici e della QKD pone fine a questo problema in maniera definitiva. Tutti i protocolli di "Quantum Key Distribution" permettono di ottenere uno scambio di chiave privata veramente segreta, ed è un qualcosa di straordinario e impossibile da ottenere senza sfruttare le proprietà della meccanica quantistica.
In questo studio si è analizzato il protocollo E91 di QKD, che permette di distribuire una chiave privata sfruttando le proprietà dell'entanglement. Come osservato dai dati ottenuti su computer quantistici reali, però, la tecnologia non è ancora così avanzata da permettere il mantenimento dello stato di un qubit e la sua misura in maniera sufficientemente precisa; le chiavi condivise con il protocollo E91 presentano quindi sempre degli errori. Su Lagos (il computer più preciso dei due su cui sono stati raccolti i dati) le chiavi condivise hanno un errore medio percentuale di 1.01 ± 2.03, 2.97 ± 2.07, 3.40 ± 1.02, 3.86 ± 0.90, per $N = 100, 500, 1000, 2000$, rispettivamente. L'altro computer utilizzato, Jakarta, invece presenta degli errori percentuali di 3.33 ± 3.61, 6.96 ± 2.83, 5.47 ± 0.96, 4.94 ± 1.01.
Dall'altro lato, l'eventuale intervento di un utente esterno (Eve) che cerca di scoprire la chiave condivisa viene sempre individuato tramite il calcolo del valore S. Per $n \geq 500$ i valori di $|S|$ misurati sono sempre maggiori di 2 in assenza di Eve e sono sempre minori di 2 in presenza del suo intervento, sia su Lagos che su Jakarta. Inoltre, l'utilizzo di due basi diverse per ottenere i bit della chiave, scelte casualmente, impedisce a Eve di misurare i qubit nella base corretta e quindi di ottenere l'intera chiave privata.

Gli attuali problemi di precisione verranno risolti in futuro con il miglioramento della tecnologia dei qubit e dei computer quantistici, e grazie anche alla possibile introduzione di una "rete quantistica" che permetta lo scambio di informazioni tra computer distanti tra loro, algoritmi di QKD potranno essere implementati con successo.

Appendice A

Progamma E91.py utilizzato sul simulatore

```
import random
from datetime import datetime
from typing import List
from qiskit import QuantumCircuit, execute, IBMQ
from qiskit.providers import Backend
from qiskit.providers.aer import QasmSimulator

# X gate
def A1(circ: QuantumCircuit):
    circ.h(0)

# Corrisponde a Z + X / sqrt(2)
def A2(circ: QuantumCircuit):
    circ.s(0)
    circ.h(0)
    circ.t(0)
    circ.h(0)

# Z gate
def A3(circ: QuantumCircuit):
    return #Nessuna operazione richiesta per la direzione Z

# Corrisponde a Z + X / sqrt(2)
def B1(circ: QuantumCircuit):
    circ.s(1)
    circ.h(1)
    circ.t(1)
    circ.h(1)

# Z
def B2(circ: QuantumCircuit):
    return  # Nessuna operazione richiesta per la direzione Z

# Corrisponde a Z - X / sqrt(2)
def B3(circ: QuantumCircuit):
    circ.s(1)
    circ.h(1)
    circ.tdg(1)
```

```
    circ.h(1)

def useIBMBackend(backendName: str):
    IBMQ.load_account()
    provider = IBMQ.get_provider(group='uni-milano-bicoc-1') # Alcuni
    ↪  backend premium sono presenti solo su questo gruppo
    backend = provider.get_backend(backendName)
    return backend

# Lo stato inziale sarà |01> - |10> / sqrt(2)
def newEntangledCircuit():
    newCircuit = QuantumCircuit(3, 3) #Il terzo qubit serve per
    ↪  implementare la strategia 3 di Eve
    newCircuit.x(0)
    newCircuit.x(1)
    newCircuit.h(0)
    newCircuit.cnot(0, 1)
    return newCircuit

def getResultsFromCircuits(circuits: List[QuantumCircuit], backend:
↪  Backend):
    job = execute(circuits, backend, shots=1)
    print(job.status())
    # Grab results from the job
    results = job.result()
    print(job.status())
    #print("Data e ora Job: ", job.creation_date()) # Il metodo
    ↪  creation_date() esiste solo sui backend dei computer reali
    print("Data e ora completamento: ", datetime.now())
    return results

def indexForResult(result: str) -> int:
    result = result[1:3] #I risultati sono al contrario, il terzo bit
    ↪  quindi è quello di Alice, il secondo quello di Bob, il terzo
    ↪  quello di Eve
    if result == '00':
        return 0
    elif result == '01':
        return 1
    elif result == '10':
        return 2
    elif result == '11':
        return 3

def eveAction1(circuits: list):
    # Strategia 1: Misuro tutti i qubit di Bob
    for circuit in circuits:
        circuit.measure(1, 1)
```

```
def eveAction2(circuits: list):
    # Strategia 2: Misuro tutti i qubit di Bob usando una direzione
    ↪ casuale tra quelle a disposizione di Bob
    for circuit in circuits:
        eve_basis = random.choice([1, 2, 3])
        if eve_basis == 1:
            B1(circuit)
        elif eve_basis == 2:
            B2(circuit)
        elif eve_basis == 3:
            B3(circuit)
        circuit.measure(1, 1)

def eveAction3(circuits: list):
    # Strategia 3: Metto in entanglement un terzo qubit posseduto da
    ↪ Eve con il qubit di Bob
    for circuit in circuits:
        circuit.cnot(1, 2)

######### Parametri configurabili - Inizio #############

_backend = QasmSimulator() # Aer's qasm_simulator
#_backend = useIBMBackend('ibm_lagos')


# Numero di coppie di qubit in entanglement generate
n=500

# Strategia di Eve; i valori possibili sono 1, 2, 3
# 0 per non avere l'intervento di Eve
eveStrategy = 0
######### Parametri configurabili - Fine #############


alice_basis = []
bob_basis = []

qubits_circuits = []

for i in range(0, n):
    #Charlie (o Alice) genera n coppie di qubit in entanglement
    qubits_circuits.append(newEntangledCircuit())

    #Alice e Bob scelgono indipendentemente una delle 3 basi
    ↪ previste per loro con cui misurare i propri qubit
    alice_basis.append(random.choice([1, 2, 3]))
    bob_basis.append(random.choice([1, 2, 3]))

####################################
```

```
# L'azione di Eve si inserisce qui #
####################################
if eveStrategy == 1:
    eveAction1(qubits_circuits)
elif eveStrategy == 2:
    eveAction2(qubits_circuits)
elif eveStrategy == 3:
    eveAction3(qubits_circuits)

#Conteggi per 00, 01, 10, 11, rispettivamente
countA1B1 = [0, 0, 0, 0] # XW observable
countA1B3 = [0, 0, 0, 0] # XV observable
countA3B1 = [0, 0, 0, 0] # ZW observable
countA3B3 = [0, 0, 0, 0] # ZV observable

aliceKey = []
bobKey = []

for i, circ in enumerate(qubits_circuits):
    if alice_basis[i] == 3 and bob_basis[i] == 2: #Verranno usati per
    ↪  la chiave
        A3(circ)
        B2(circ)
    elif alice_basis[i] == 2 and bob_basis[i] == 1: #Verranno usati
    ↪  per la chiave
        A2(circ)
        B1(circ)
    elif alice_basis[i] == 1 and bob_basis[i] == 3:
        A1(circ)
        B3(circ)
    elif alice_basis[i] == 1 and bob_basis[i] == 1:
        A1(circ)
        B1(circ)
    elif alice_basis[i] == 3 and bob_basis[i] == 1:
        A3(circ)
        B1(circ)
    elif alice_basis[i] == 3 and bob_basis[i] == 3:
        A3(circ)
        B3(circ)

    circ.measure(0, 0)
    circ.measure(1, 1)
    circ.measure(2, 2)


backendResults = getResultsFromCircuits(qubits_circuits, _backend)

for i, result in enumerate(backendResults.get_counts()):
    resultBits = list(result.keys())[0]
```

```
    if (alice_basis[i] == 3 and bob_basis[i] == 2) or (alice_basis[i]
    ↪  == 2 and bob_basis[i] == 1):
        aliceKey.append(resultBits[2])
        bobKey.append(resultBits[1])
    elif alice_basis[i] == 1 and bob_basis[i] == 3:
        j = indexForResult(resultBits)
        countA1B3[j] += 1
    elif alice_basis[i] == 1 and bob_basis[i] == 1:
        j = indexForResult(resultBits)
        countA1B1[j] += 1
    elif alice_basis[i] == 3 and bob_basis[i] == 1:
        j = indexForResult(resultBits)
        countA3B1[j] += 1
    elif alice_basis[i] == 3 and bob_basis[i] == 3:
        j = indexForResult(resultBits)
        countA3B3[j] += 1

aliceKeyString = ""
bobKeyString = ""
errorCount = 0

for (i, aliceBitString) in enumerate(aliceKey):
    bobBitString = bobKey[i]
    bobBitString = str(int(not bool(int(bobBitString)))) #applico NOT
    aliceKeyString += aliceBitString
    bobKeyString += bobBitString
    if aliceBitString != bobBitString:
        errorCount += 1



# number of the results obtained from the measurements in a
↪  particular basis
total11 = sum(countA1B1)
total13 = sum(countA1B3)
total31 = sum(countA3B1)
total33 = sum(countA3B3)

# Valori d'aspettazione
expect11 = (countA1B1[0] - countA1B1[1] - countA1B1[2] +
↪  countA1B1[3]) / total11  # -1/sqrt(2)
expect13 = (countA1B3[0] - countA1B3[1] - countA1B3[2] +
↪  countA1B3[3]) / total13  # 1/sqrt(2)
expect31 = (countA3B1[0] - countA3B1[1] - countA3B1[2] +
↪  countA3B1[3]) / total31  # -1/sqrt(2)
expect33 = (countA3B3[0] - countA3B3[1] - countA3B3[2] +
↪  countA3B3[3]) / total33  # -1/sqrt(2)

# Valore di correlazione CHSH
```

```
corr = expect11 - expect13 + expect31 + expect33

# CHSH inequality test
print('----------------------------------------')
print('----------------Risultati---------------')
print('----------------------------------------')
print('CHSH correlation value: ' + str(round(abs(corr), 5)))
print('The shared key is: ' + aliceKeyString)
print('Key length: ' + str(len(aliceKey)))
print('Number of errors in key shared: ' + str(errorCount))
```

Appendice B

Programma ottimizzato per computer reali

```
import random
from datetime import datetime
from typing import List
from qiskit import QuantumCircuit, execute, IBMQ
from qiskit.providers import Backend

# X gate
def A1(circ: QuantumCircuit):
    circ.h(0)

# Corrisponde a Z + X / sqrt(2)
def A2(circ: QuantumCircuit):
    circ.s(0)
    circ.h(0)
    circ.t(0)
    circ.h(0)

# Z gate
def A3(circ: QuantumCircuit):
    return #Nessuna operazione richiesta per la direzione Z


# Corrisponde a Z + X / sqrt(2)
def B1(circ: QuantumCircuit):
    circ.s(1)
    circ.h(1)
    circ.t(1)
    circ.h(1)

# Z
def B2(circ: QuantumCircuit):
    return  # Nessuna operazione richiesta per la direzione Z

# Corrisponde a Z - X / sqrt(2)
def B3(circ: QuantumCircuit):
    circ.s(1)
    circ.h(1)
    circ.tdg(1)
```

```
    circ.h(1)

def useIBMBackend(backendName: str):
    IBMQ.load_account()
    provider = IBMQ.get_provider(group='uni-milano-bicoc-1') # Alcuni
    ↪  backend premium sono presenti solo su questo gruppo
    backend = provider.get_backend(backendName)
    return backend

# Lo stato inziale sarà |01> - |10> / sqrt(2)
def newEntangledCircuit():
    newCircuit = QuantumCircuit(3, 3) #Il terzo qubit serve per
    ↪  implementare la strategia 3 di Eve
    newCircuit.x(0)
    newCircuit.x(1)
    newCircuit.h(0)
    newCircuit.cnot(0, 1)
    return newCircuit


def getResultsFromCircuits(circuits: List[QuantumCircuit], backend:
↪  Backend):
    totalResults = []
    jobsExecuted = []
    #Aggiungo prima tutti i job in coda, poi aspetto i risultati di
    ↪  tutti
    for idx, circuit in enumerate(circuits):
        jobShots = shots[idx]
        job = execute(circuit, backend, shots=jobShots)
        print(job.status())
        jobsExecuted.append(job)

    for job in jobsExecuted:
        results = job.result()
        print(job.status())
        #print("Data e ora Job: ", job.creation_date()) # Il metodo
        ↪  creation_date() esiste solo sui backend dei computer
        ↪  reali
        print("Data e ora completamento: ", datetime.now())
        totalResults.append(results.get_counts())

    return totalResults


def indexForResult(result: str) -> int:
    result = result[1:3] #I risultati sono al contrario, il terzo bit
    ↪  quindi è quello di Alice, il secondo quello di Bob, il terzo
    ↪  quello di Eve
    if result == '00':
```

```
        return 0
    elif result == '01':
        return 1
    elif result == '10':
        return 2
    elif result == '11':
        return 3

def eveAction1(circuit: QuantumCircuit):
    # Strategia 1: Misuro tutti i qubit di Bob
    circuit.measure(1, 1)

def eveAction3(circuit: QuantumCircuit):
    # Strategia 3: Metto in entanglement un terzo qubit posseduto da
    ↪  Eve con il qubit di Bob
    circuit.cnot(1, 2)

######### Parametri configurabili - Inizio #############

#_backend = QasmSimulator() # Aer's qasm_simulator
_backend = useIBMBackend('ibm_lagos')

# Numero di coppie di qubit in entanglement generate
n=1000

# Strategia di Eve; i valori possibili sono 1, 2, 3
# 0 per non avere l'intervento di Eve
eveStrategy = 0
######### Parametri configurabili - Fine #############


alice_basis = []
bob_basis = []

#Sono solo 9 le possibili combinazioni, senza l'intervento di Eve
qubits_circuits = []
for i in range(9):
    circ = newEntangledCircuit()

    ####################################
    # L'azione di Eve si inserisce qui #
    ####################################
    if eveStrategy == 1:
        eveAction1(circ)
    #elif eveStrategy == 2: #La strategia 2 non è implementabile con
    ↪  solo 9 circuiti pre-impostati
    #    eveAction2(circ)
    elif eveStrategy == 3:
        eveAction3(circ)
```

```
        if i == 0:
            A1(circ)
            B1(circ)
        elif i == 1:
            A1(circ)
            B2(circ)
        elif i == 2:
            A1(circ)
            B3(circ)
        elif i == 3:
            A2(circ)
            B1(circ)
        elif i == 4:
            A2(circ)
            B2(circ)
        elif i == 5:
            A2(circ)
            B3(circ)
        elif i == 6:
            A3(circ)
            B1(circ)
        elif i == 7:
            A3(circ)
            B2(circ)
        elif i == 8:
            A3(circ)
            B3(circ)

        circ.measure(0, 0)
        circ.measure(1, 1)
        circ.measure(2, 2)

        qubits_circuits.append(circ)

    # Charlie (o Alice) genera n coppie di qubit in entanglement
    # Alice e Bob scelgono indipendentemente una delle 3 basi previste
    ↪  per loro con cui misurare i propri qubit
    # In totale sono 9 combinazioni possibili
    # Distribuisco i conteggi (shots) casualmente sui 9 circuiti
    shots = [0, 0, 0, 0, 0, 0, 0, 0, 0]
    for i in range(0, n):
        randomCircuitIndex = random.choice([0, 1, 2, 3, 4, 5, 6, 7, 8])
        shots[randomCircuitIndex] += 1

    #Conteggi per 00, 01, 10, 11, rispettivamente
    countA1B1 = [0, 0, 0, 0] # XW observable
    countA1B3 = [0, 0, 0, 0] # XV observable
    countA3B1 = [0, 0, 0, 0] # ZW observable
```

```
countA3B3 = [0, 0, 0, 0] # ZV observable


aliceKey = []
bobKey = []

backendResults = getResultsFromCircuits(qubits_circuits, _backend)

for i, circuitResult in enumerate(backendResults):
    # Le basi utilizzabili come chiave
    if i == 3 or i == 7:
        for resultBits in circuitResult.keys():
            for j in range(circuitResult[resultBits]):
                aliceKey.append(resultBits[2])
                bobKey.append(resultBits[1])
    else:
        for resultBits in circuitResult.keys():
            k = indexForResult(resultBits)
            if i == 0:
                countA1B1[k] += circuitResult[resultBits]
            elif i == 2:
                countA1B3[k] += circuitResult[resultBits]
            elif i == 6:
                countA3B1[k] += circuitResult[resultBits]
            elif i == 8:
                countA3B3[k] += circuitResult[resultBits]


aliceKeyString = ""
bobKeyString = ""
errorCount = 0

for (i, aliceBitString) in enumerate(aliceKey):
    bobBitString = bobKey[i]
    bobBitString = str(int(not bool(int(bobBitString)))) #applico NOT
    aliceKeyString += aliceBitString
    bobKeyString += bobBitString
    if aliceBitString != bobBitString:
        errorCount += 1

# Totale risultati ottenuti per ogni base
total11 = sum(countA1B1)
total13 = sum(countA1B3)
total31 = sum(countA3B1)
total33 = sum(countA3B3)

# Valori d'aspettazione per ogni base
expect11 = (countA1B1[0] - countA1B1[1] - countA1B1[2] +
↪  countA1B1[3]) / total11  # -1/sqrt(2)
```

```
expect13 = (countA1B3[0] - countA1B3[1] - countA1B3[2] +
↪  countA1B3[3]) / total13  # 1/sqrt(2)
expect31 = (countA3B1[0] - countA3B1[1] - countA3B1[2] +
↪  countA3B1[3]) / total31  # -1/sqrt(2)
expect33 = (countA3B3[0] - countA3B3[1] - countA3B3[2] +
↪  countA3B3[3]) / total33  # -1/sqrt(2)

# Valore di correlazione CHSH
corr = expect11 - expect13 + expect31 + expect33

# CHSH inequality test
print('----------------------------------------')
print('---------------Risultati----------------')
print('----------------------------------------')
print('CHSH correlation value: ' + str(round(abs(corr), 5)))
print('The shared key is: ' + aliceKeyString)
print('Key length: ' + str(len(aliceKey)))
print('Number of errors in key shared: ' + str(errorCount))
```

Bibliografia

[1] Charles H. Bennett e Gilles Brassard. «Quantum cryptography: Public key distribution and coin tossing». In: *Theoretical Computer Science* 560 (2014, original 1984). Theoretical Aspects of Quantum Cryptography – celebrating 30 years of BB84, pp. 7–11. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2014.05.025. URL: https://www.sciencedirect.com/science/article/pii/S0304397514004241 (cit. a p. 44).

[2] B.S. Cirel'son. «Quantum generalizations of Bell's inequality». In: *Lett Math Phys* 4 (1980), pp. 93–100. URL: https://doi.org/10.1007/BF00417500 (cit. a p. 5).

[3] John F. Clauser et al. «Proposed Experiment to Test Local Hidden-Variable Theories». In: *Phys. Rev. Lett.* 23 (15 ott. 1969), pp. 880–884. DOI: 10.1103/PhysRevLett.23.880. URL: https://link.aps.org/doi/10.1103/PhysRevLett.23.880 (cit. a p. 6).

[4] Artur K. Ekert. «Quantum cryptography based on Bell's theorem». In: *Phys. Rev. Lett.* 67 (6 ago. 1991), pp. 661–663. DOI: 10.1103/PhysRevLett.67.661. URL: https://link.aps.org/doi/10.1103/PhysRevLett.67.661 (cit. a p. 4).

[5] Lov K. Grover. «A fast quantum mechanical algorithm for database search». In: (1996). DOI: 10.48550/ARXIV.QUANT-PH/9605043. URL: https://arxiv.org/abs/quant-ph/9605043 (cit. a p. 2).

[6] IBM. *Qiskit documentation v0.36*. Last accessed 15/02/2023. 2022. URL: https://qiskit.org/documentation/stable/0.36/index.html (cit. a p. 7).

[7] Gregor Weihs et al. «Violation of Bell's Inequality under Strict Einstein Locality Conditions». In: *Physical Review Letters* 81.23 (dic. 1998), pp. 5039–5043. DOI: 10.1103/physrevlett.81.5039. URL: https://doi.org/10.1103%5C%2Fphysrevlett.81.5039 (cit. a p. 4).

[8] W. Wootters e W. Zurek. «A single quantum cannot be cloned». In: *Nature* 299 (ago. 1982), pp. 802–803. DOI: 10.1038/299802a0. URL: https://www.nature.com/articles/299802a0 (cit. a p. 24).

Ringraziamenti

Ringrazio i miei relatori - Dr. Andrea Giachero e Dr. Danilo Labranca - per avermi introdotto in questo mondo del quantum computing e per aver avuto la pazienza di seguirmi fino alla fine di questo percorso.

Ringrazio la mia famiglia che mi ha sempre incoraggiato e mi ha trasmesso la forza di andare avanti.

Ringrazio i miei amici che mi supportano e sopportano da anni... la lista sarebbe troppo lunga; chi mi ha offerto affetto e parole d'incoraggiamento lo sa, e gli sono infinitamente riconoscente.

Un pezzo di questo percorso si è concluso.
To be continued...