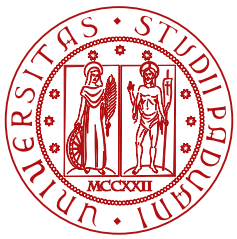


UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Sviluppo di software per quantum key distribution

Relatore
Prof. Vallone Giuseppe

Laureando
Rubin Marco

Correlatore
Ing. Berra Federico

ANNO ACCADEMICO 2022-2023

Data di laurea 28/09/2023

Ai miei familiari e agli amici che mi hanno sopportato in questi anni

Sommario

L'obiettivo di questa tesi riguarda lo sviluppo di software nel linguaggio Python per la gestione di un sistema di *quantum key distribution* basato su un *system-on-a-chip*. Sfruttando le proprietà quantistiche della luce, la *quantum key distribution* permette di generare e condividere in modo sicuro una chiave crittografica simmetrica. Vengono quindi descritti i programmi per l'invio e la ricezione dei dati attraverso una connessione Ethernet, e le analisi volte a verificare l'affidabilità del sistema. In conclusione, viene proposta una riflessione sullo stato attuale del sistema, e sulla sua possibile adozione al di fuori dei laboratori di ricerca.

Indice

Elenco delle figure

Capitolo 1

Introduzione

1.1 Comunicazioni cifrate

Le comunicazioni cifrate pervadono la società contemporanea, dalle transazioni bancarie alle applicazioni di messaggistica istantanea. Nascono dalla necessità che il contenuto di un messaggio sia conoscibile alle sole parti interessate, da cui l'importanza di studiare e raffinare tecniche di cifratura diverse per poterne garantire la segretezza. Al centro di queste tecniche si trova la chiave crittografica, usata per cifrare e decifrare i messaggi scambiati. Oltre alle problematiche legate al trasporto dei messaggi in un canale di comunicazione, bisogna preoccuparsi che solo i partecipanti di una conversazione abbiano accesso alla chiave: dovendo in principio restare segreta, non conviene scambiarla attraverso lo stesso canale dei messaggi.

Negli ultimi decenni si è notato che alcuni risultati della meccanica quantistica sono particolarmente adatti a questo scopo, e sono stati realizzati dei protocolli per la creazione e condivisione delle chiavi crittografiche, che garantiscono una sicurezza incondizionata della chiave. Così è nata una branca della crittografia che applica i principi della meccanica quantistica alla distribuzione delle chiavi, la *quantum key distribution*.

1.2 Fondamenti di *quantum key distribution* (QKD)

Nei protocolli di *quantum key distribution*, la chiave crittografica viene trasmessa in un canale di comunicazione quantistico, cioè in cui è possibile sfruttare le proprietà quantistiche dei sistemi fisici coinvolti. Il trasmettitore, "Alice", codifica l'informazione nello stato quantistico di un oggetto fisico, che viene poi inviato al ricevitore, "Bob". I mezzi più comuni sono i fotoni, perché sono molto facili da produrre e da trasportare, grazie alla loro bassa interazione con l'ambiente circostante. L'informazione viene tipicamente codificata nella loro polarizzazione, che determina la direzione del campo elettrico di un'onda elettromagnetica piana: quando si misura

la polarizzazione di un fotone, la si trova in uno solo tra due stati, o verticale oppure orizzontale (analogamente, cambiando sistema di riferimento, o diagonale o antidiagonale). A ciascuno dei due possibili risultati si può associare un valore, per esempio "0" e "1": così si stabilisce un collegamento coi bit della teoria dell'informazione. Ma la QKD sfrutta le proprietà della polarizzazione *prima* che venga misurata, quando è descrivibile solo dalle leggi della meccanica quantistica.

Di seguito sono presentati alcuni principi della meccanica quantistica che soggiacciono ai principali protocolli, cui segue una discussione del protocollo BB84. Per una descrizione più dettagliata, si possono consultare le sezioni 2.4, 3.1 e 4.1-4.3 di *Principles of Quantum Computation and Information* di Giuliano Benenti *et al*.[1]

1.2.1 Misure e sovrapposizioni

A differenza della meccanica classica, la meccanica quantistica è strettamente legata alla statistica, perché predice le probabilità con cui i risultati si verificano. Come accennato in precedenza, se si vuole misurare la polarizzazione di un fotone, questa risulterà, per esempio, o verticale oppure orizzontale, con certe probabilità.

Secondo i principi della meccanica quantistica, lo stato della grandezza fisica di interesse è descritto da un vettore di uno spazio di Hilbert complesso. Prima di una misura è esprimibile come combinazione lineare (o "sovrapposizione") dei vettori di una base, ciascuno associato a un risultato della misura. Ad esempio, data la base

$$\begin{aligned} |0\rangle &= (1,0) \\ |1\rangle &= (0,1) \end{aligned} \tag{1.1}$$

si può descrivere una generica polarizzazione come:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1.2}$$

Quando viene misurato, lo stato "collassa" in uno dei vettori della base, e la probabilità di ottenere quel risultato è data dal quadrato del modulo del coefficiente associato, detto "ampiezza di probabilità". In questo caso si ha:

$$\begin{aligned} P(0) &= |\alpha|^2 \\ P(1) &= |\beta|^2 \end{aligned} \tag{1.3}$$

Poiché i quadrati dei moduli dei coefficienti sono delle probabilità, la loro somma deve essere la probabilità totale, e perciò deve essere pari a 1:

$$|\alpha|^2 + |\beta|^2 = 1 \tag{1.4}$$

Sistemi quantistici di questo tipo, descritti da stati bidimensionali, possono essere considerati dei bit quantistici, o "qubit".

1.2.2 Teorema della non clonazione

Un risultato importante per la crittografia quantistica è il teorema della non clonazione, che garantisce l'impossibilità di clonare uno stato arbitrario $|\psi\rangle$ in un altro stato $|\varphi\rangle$. Questo teorema introduce una notevole difficoltà per una spia, poiché le impedisce di copiare un messaggio che vuole intercettare in un canale quantistico.

Per vedere come non sia possibile clonare uno stato arbitrario, si può considerare uno stato $|\psi\rangle$, descritto dall'equazione 1.2, e uno stato temporaneo $|\varphi\rangle$ su cui copiare $|\psi\rangle$. Considerati congiuntamente, lo stato composto $|\psi, \varphi\rangle$ è uno stato quantistico a sua volta, ovvero è soggetto alle stesse regole di $|\psi\rangle$ e $|\varphi\rangle$, ed è descritto dal prodotto tensoriale $|\psi\rangle \otimes |\varphi\rangle$. Una macchina capace di clonare $|\psi\rangle$ dovrebbe trasformare $|\varphi\rangle$ in $|\psi\rangle$, senza modificare lo stato $|\psi\rangle$, ovvero dovrebbe implementare la seguente operazione O:

$$O\,|\psi, \varphi\rangle = |\psi, \psi\rangle \tag{1.5}$$

Perché la clonazione sia corretta, le ampiezze α e β di $|\psi\rangle$ devono rimanere invariate, perciò si richiede che:

$$|\psi, \psi\rangle \stackrel{!}{=} \alpha\,|00\rangle + \beta\,|11\rangle \tag{1.6}$$

Tuttavia,

$$\begin{aligned} |\psi, \psi\rangle &= |\psi\rangle \otimes |\psi\rangle \\ &= (\alpha\,|0\rangle + \beta\,|1\rangle) \otimes (\alpha\,|0\rangle + \beta\,|1\rangle) \\ &= \alpha^2\,|00\rangle + \alpha\beta\,|01\rangle + \alpha\beta\,|10\rangle + \beta^2\,|11\rangle \end{aligned} \tag{1.7}$$

Nell'ultimo passaggio si è sfruttata la linearità del prodotto tensoriale.

Le due espressioni 1.6 e 1.7 sono generalmente diverse, per questo non è possibile realizzare una macchina capace di clonare uno stato arbitrario. Quando invece lo stato è conosciuto, ad esempio quando corrisponde ad uno dei bit $|0\rangle$ o $|1\rangle$, è certamente possibile clonarlo, per questo è possibile copiare i messaggi classici.

1.2.3 Il protocollo BB84

Il primo protocollo di QKD è stato pubblicato da Charles Bennett e Gilles Brassard nel 1984,[2] per questo è chiamato anche "BB84". È stato proposto per mostrare come i principi della meccanica quantistica possono garantire la segretezza di una comunicazione, a differenza di altre tecniche basate su problemi aperti della matematica, come accade per i sistemi di crittografia asimmetrica.

Questo protocollo prevede che due parti, Alice e Bob, si scambino dei messaggi attraverso un canale pubblico con autenticazione, e la chiave crittografica tramite un canale quantistico. L'autenticazione è necessaria per evitare che una spia, Eve, possa impersonarsi come Alice o Bob. Del canale quantistico, invece, vengono sfruttate le proprietà descritte in 1.2.1 e 1.2.2. La chiave viene codificata nella polarizzazione dei fotoni, che possono essere preparati in uno degli stati

$$\begin{aligned} |H\rangle &= (1,0) \\ |V\rangle &= (0,1) \\ |D\rangle &= \frac{1}{\sqrt{2}}|H\rangle + \frac{1}{\sqrt{2}}|V\rangle \\ |A\rangle &= \frac{1}{\sqrt{2}}|H\rangle - \frac{1}{\sqrt{2}}|V\rangle \end{aligned} \tag{1.8}$$

e misurati in una delle basi

$$\begin{aligned} + &= \{|H\rangle, |V\rangle\} \\ \times &= \{|D\rangle, |A\rangle\} \end{aligned} \tag{1.9}$$

Inizialmente, Alice sceglie una sequenza di bit e una sequenza di basi ($+$ o $\times$) casuali della stessa lunghezza. Poi invia a Bob la sequenza di bit tramite i fotoni, codificando il bit 0 in $|H\rangle$ e $|D\rangle$ e il bit 1 in $|V\rangle$ e $|A\rangle$, secondo la base che corrisponde al bit da inviare. Bob, non conoscendo lo stato dei fotoni, quando ne riceve uno lo misura in una delle due basi scelta a caso, e interpreta il risultato come un bit 0 o 1.

La conversazione riprende poi nel canale classico, in cui Bob invia ad Alice la lista di basi usate per misurare i fotoni ricevuti, e Alice risponde indicando quali basi combaciano. I due poi scartano i bit per cui le basi non combaciano.

Se Eve si intromettesse durante la trasmissione della chiave, non potendo clonare i fotoni per via del teorema della non clonazione, sarebbe costretta a misurarli in una delle due basi, scelta casualmente, e inviarne di nuovi a Bob. Così facendo cambierebbe la statistica dei bit di Bob, perché introdurrebbe degli errori dovuti alle scelte casuali delle basi, e verrebbe scoperta. I due allora potrebbero decidere di scartare la chiave e inviarne una nuova. Per assicurarsi dell'assen-

Basi di Alice	×	+	+	×	+	+	×	×
Bit di Alice	1	1	0	0	1	0	1	0
Basi di Bob	+	+	×	×	+	×	+	×
Bit di Bob	1	1	1	0	1	0	1	0
Chiave		1		0	1			0

Tabella 1.1: Esempio di un'applicazione del protocollo BB84.

za di intromissioni, Alice e Bob possono scambiarsi alcuni bit della chiave per verificare che combacino, al costo di perdere la segretezza di quei bit. In caso di esito positivo, i due possono essere sicuri di non essere stati sufficientemente spiati, e possono usare i bit ottenuti come un *one-time pad* (OTP) per comunicare nel canale pubblico. Quando la chiave viene consumata, si ripete il processo per inviarne una nuova.

1.3 Tecnologie per la QKD

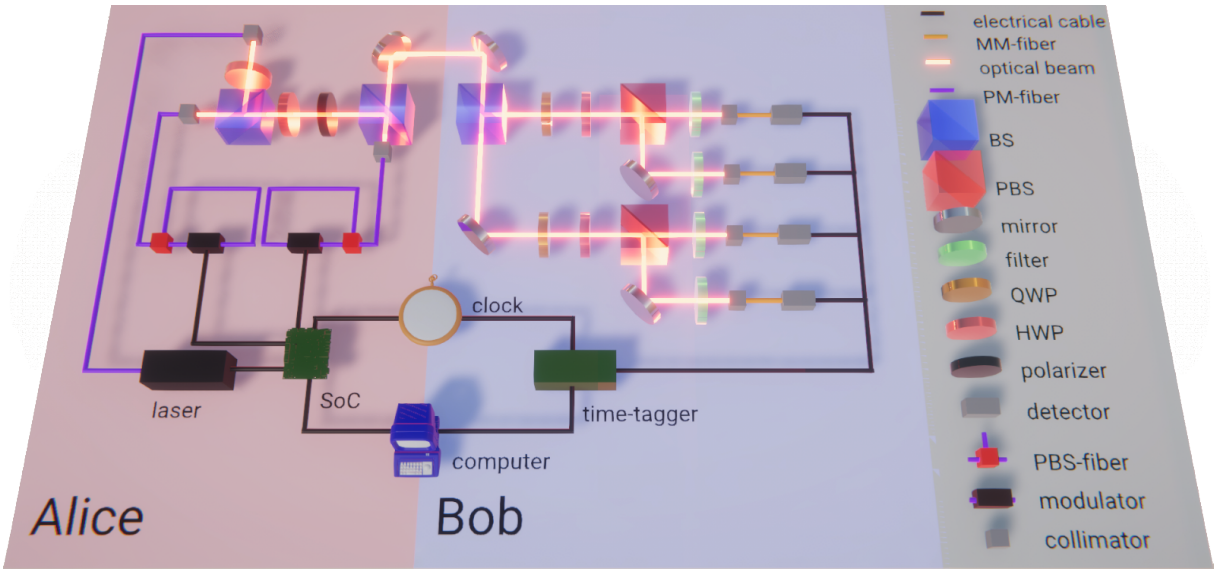


Figura 1.1: Schema del sistema di QKD progettato dal gruppo QuantumFuture.

Un sistema di QKD trasmette come segnali degli stati quantistici ben definiti, in questo caso dei qubit, e dev'essere costruito in modo da non introdurre ulteriori correlazioni tra i qubit prodotti, né gradi di libertà che possano essere sfruttati da malintenzionati per spiare la comunicazione. Ad oggi, le maggiori difficoltà per l'implementazione di questi sistemi sono poste dall'attenuazione del segnale nel mezzo di propagazione, generalmente l'aria ("*free space*") o la fibra ottica, che potrebbe portare a un tasso d'errore molto alto, vanificando così gli sforzi della trasmissione. Le comunicazioni in *free space* sono preferibili a quelle in fibra ottica perché, in

assenza di ripetitori, grazie ad appositi satelliti è possibile coprire maggiori distanze rispetto alla fibra ottica.

Ad oggi non è ancora chiaro quale sia il sistema ottimale per la comunicazione satellitare, ecco perché Federico Berra *et al.* del gruppo di ricerca QuantumFuture dell'Università degli Studi di Padova[3] hanno progettato un sistema di QKD per le comunicazioni in *free space*, che opera nella banda elettromagnetica vicina agli infrarossi (*near-infrared*, NIR), attorno agli $800\,\text{nm}$.[4] Il sistema è composto da un trasmettitore, Alice, e un ricevitore, Bob, entrambi collegati ad un computer e sincronizzati dallo stesso *clock*. Il suo schema è mostrato in figura 1.1.

Alice è un trasmettitore elettro-ottico in grado di generare un treno di impulsi di fotoni mediante l'utilizzo di un laser, impulsato ad una frequenza di $50\,\text{MHz}$. Tramite dei collegamenti misti, sia in *free space* che in fibra ottica, questi passano attraverso un modulatore dell'intensità e uno della polarizzazione, e infine vengono inviati al ricevitore attraverso un canale in *free space*. L'intensità degli impulsi può essere "alta" o "bassa", mentre la polarizzazione dei fotoni può essere $|H\rangle$, $|V\rangle$ o $|D\rangle$. Sia il laser che i modulatori sono controllati da un *system-on-a-chip* (SoC), che integra in una scheda dedicata un *field-programmable gate array* (FPGA) per modulare l'impulso dei fotoni, e una CPU che permette al computer di controllare la modulazione.

Entrambi i modulatori sono basati sullo schema iPOGNAC,[5] che permette di modulare la polarizzazione dei fotoni senza una fase di calibrazione, né al trasmettitore, né al ricevitore, con un basso tasso d'errore. In particolare, il modulatore dell'intensità è presente per implementare il protocollo BB84 con stati "esca" (*decoy*), proposto da Won-Young Hwang nel 2003.[6] La modifica al protocollo originale consiste nel trasmettere degli impulsi con più fotoni invece di uno solo, alcuni dei quali fungono da esche per Eve. La quantità dei fotoni è descritta da una distribuzione di Poisson con $\lambda < 1$. Al termine della trasmissione, Alice comunica a Bob anche le intensità per ciascun qubit inviato, e in base alla differenza rispetto alle intensità registrate da Bob è possibile determinare una intromissione da parte di Eve.

Bob è invece un ricevitore opto-elettrico che converte i fotoni in impulsi elettrici tramite dei fotodiodi a singolo fotone (*single-photon avalanche diode*, SPAD), che vengono collezionati da un convertitore *time-to-digital* che ne registra gli istanti d'arrivo e li invia al computer per elaborarli.

Il sistema adottato durante il corso del progetto è basato sullo schema appena descritto. Per una descrizione più generale delle tecnologie per la QKD si può fare riferimento alla sezione 3 di *A brief introduction of quantum cryptography for engineers* di Bing Qi *et al*.[7]

Capitolo 2

Sviluppo dei software

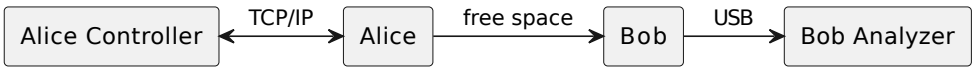


Figura 2.1: Schema del sistema di QKD.
Su ogni freccia è indicato quale collegamento rappresenta.

Il progetto di tesi è stato incentrato sullo sviluppo di software che permettessero di inviare e ricevere correttamente una chiave crittografica, sfruttando il sistema di *quantum key distribution* sviluppato da Federico Berra *et al.*, discusso nella sezione 1.3. L'apparato è descritto più in dettaglio nella sezione 3.1.

L'obiettivo principale consisteva nel trasmettere continuamente una chiave di 1024 simboli. Per farlo, si ha avuto bisogno di un programma che configurasse Alice e le inviasse la chiave da trasmettere: questo programma, *Alice Controller*, era già presente all'inizio del progetto, ma permetteva di inviare unicamente chiavi preimpostate, perciò è stato aggiornato dove necessario per poter soddisfare la richiesta. All'estremo opposto di Alice, Bob necessitava di un programma che ne acquisisse i dati immagazzinati e li mostrasse sullo schermo del computer, per poter verificare in tempo reale la correttezza della comunicazione. Queste richieste hanno portato allo sviluppo di *Bob Analyzer*.

2.1 Progettazione

2.1.1 *Alice Controller*

La versione di *Alice Controller* presente all'inizio di questo progetto permetteva d'inviare solo chiavi preimpostate, di lunghezze fissate, ovvero in modalità "*fixed*", seondo la terminologia adottata da Alice. Il programma è stato aggiornato per poter inviare continuamente una chiave di lunghezza arbitraria, in questo caso prelevata da un file, implementando la modalità

"*streaming*". In entrambe le modalità, la chiave viene trasmessa modulando la polarizzazione e l'intensità dell'impulso del laser, eventualmente ritardando i segnali rispetto al periodo dell'impulso. Il ritardo consiste nell'allineare temporalmente i segnali di modulazione di intensità e polarizzazione con i rispettivi impulsi ottici.

In figura 2.3 è presente il diagramma temporale della comunicazione tra il programma ("PC") e il trasmettitore ("Alice") in modalità *streaming*: le frecce solide indicano i comandi inviati dal PC e le risposte ricevute da Alice, scambiati nello stesso canale, mentre quelle tratteggiate indicano l'invio di dati in canali dedicati.

Questa modalità richiede che vengano inviate al trasmettitore due chiavi, una nel canale di comunicazione dedicato ai "dati di polarizzazione", l'altra ai "dati d'intensità". Inizialmente il software comunicava con Alice solo tramite il canale dedicato ai comandi, uno dei *port* esposti da Alice via TCP/IP, perciò è stato modificato per sfruttare gli altri due quando necessario. Ciò avviene quando, in figura 2.3, il PC deve inviare ad Alice una parte della chiave "P" (polarizzazione) o della chiave "I" (intensità). Come si può notare dalla figura, lo *streaming* è diviso in due fasi: prima il trasmettitore viene configurato e precarica nella sua memoria la prima parte di entrambe le chiavi; poi, quando viene avviata la QKD, questo trasmette la parte in suo possesso e ne richiede una nuova, di continuo, finché la QKD non viene terminata.

Per raggiungere gli obiettivi proposti, gli aggiornamenti apportati ad *Alice Controller* sono stati suddivisi in quattro componenti grafiche, che permettono di:

- impostare i canali di comunicazione col trasmettitore a cui collegarsi;
- selezionare i file delle due chiavi;
- selezionare la modalità d'invio delle chiavi indicate, che attiva la fase di precaricamento della modalità *streaming* quando viene selezionata la voce corrispondente;
- avviare e terminare la QKD.

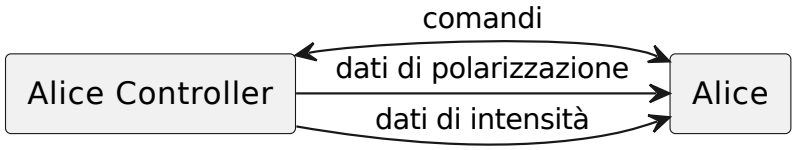


Figura 2.2: Schema dei canali di comunicazione tra *Alice Controller* e Alice.

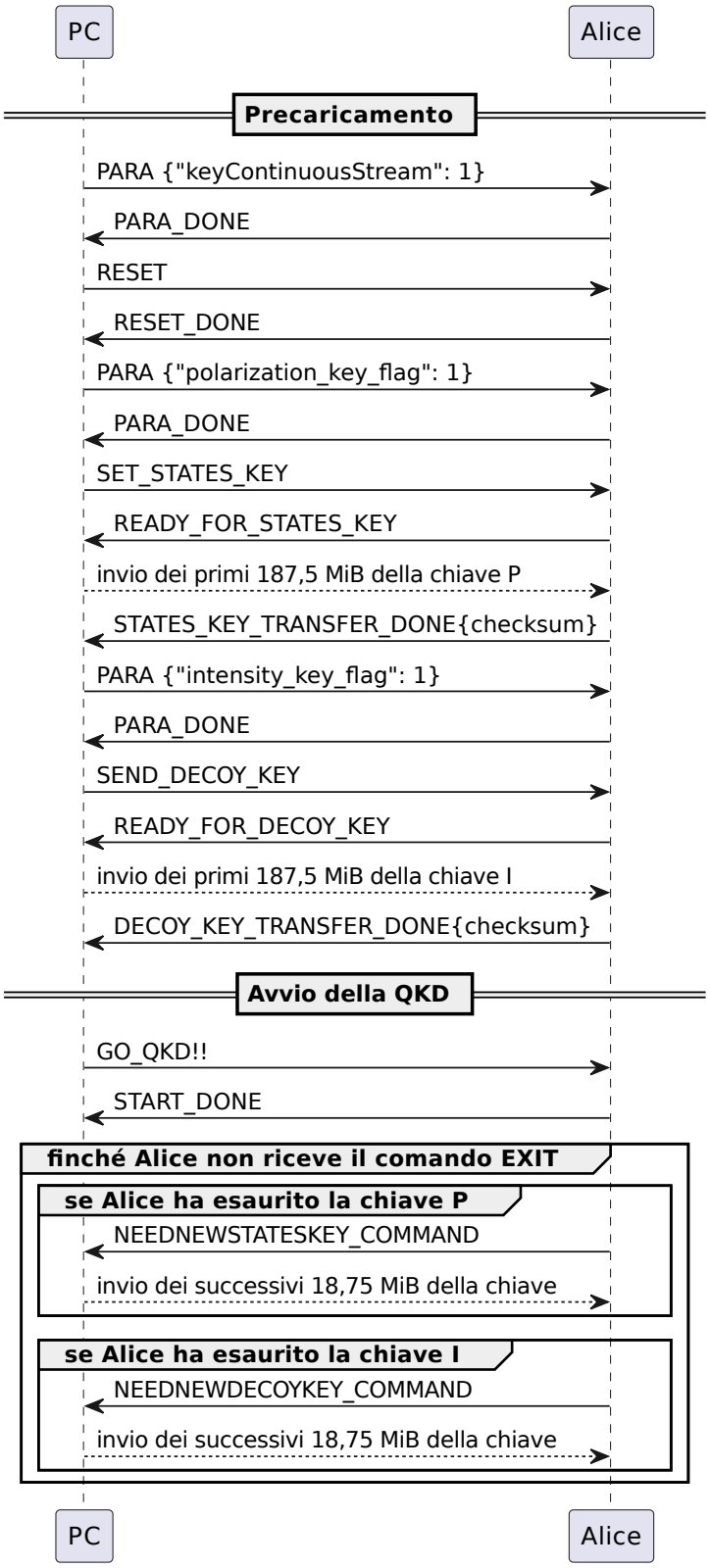


Figura 2.3: Diagramma temporale della comunicazione tra il PC e Alice in *streaming*.

2.1.2 *Bob Analyzer*

Bob Analyzer, invece, è stato realizzato interamente durante il corso del progetto. Il suo sviluppo è nato principalmente per due scopi: la visualizzazione dei dati acquisiti e il loro salvataggio sul disco.

In primis, era necessario analizzare in tempo reale i dati acquisiti per verificare il funzionamento del sistema di QKD. Fissato un periodo d'integrazione T_{int}, si è richiesto che il software mostrasse l'istogramma delle misurazioni registrate da ogni canale durante ciascun T_{int}. In particolare, sarebbe stato sufficiente visualizzarne la porzione relativa ai primi N_{hist} simboli ricevuti. Così sarebbero stati notati eventuali errori rispetto alle previsioni, e si sarebbero potuti modificare i parametri del trasmettitore tramite *Alice Controller* o riconfigurare il sistema di QKD. In alcuni di questi casi, per riflettere le modifiche effettuate al sistema, sarebbe stato necessario riconfigurare anche *Bob Analyzer*, per questo è stato richiesto di poterne cambiare alcuni parametri in tempo reale. Nello specifico, i parametri interessati sono i seguenti: il periodo impiegato dal trasmettitore a inviare un simbolo T_s, la lunghezza della chiave N_{seq}, il numero dei primi simboli della chiave di cui mostrare l'istogramma N_{hist}, il periodo d'integrazione T_{int} e la granularità dell'istogramma di ciascun simbolo, ovvero il numero di intervalli per simbolo N_{bin}.

Inoltre, per un'analisi più approfondita, si aveva bisogno di salvare automaticamente i dati acquisiti per periodi anche superiori a un'ora. Inizialmente veniva usato un software fornito col ricevitore, Daisy di qutools,[8] ma poiché è stato progettato per rapidi test di laboratorio, tipicamente di qualche minuto, si è rivelato inadeguato per gli scopi del progetto: il programma non è riuscito a funzionare per misurazioni lunghe, e i file generati dal salvataggio dei dati erano troppo pesanti. Da qui la necessità di un'alternativa più efficiente, sia in termini di spazio che di tempo, e che evitasse perdite di dati.

Per soddisfare i requisiti esposti precedentemente, la progettazione di *Bob Analyzer* lo ha visto suddiviso in quattro componenti, una di programmazione, il cui scopo è collegarsi al ricevitore e acquisirne periodicamente i dati immagazzinati, e tre grafiche, che permettono di:

- salvare sul disco i dati acquisiti, manualmente o in automatico con un timer;
- mostrare gli istogrammi richiesti;
- modificare i parametri sopra elencati.

2.2 Implementazione

Alice Controller e *Bob Analyzer* sono stati realizzati all'interno del gruppo di ricerca QuantumFuture, in un progetto mirato a uniformare lo sviluppo dei software del laboratorio: la scelta del linguaggio di programmazione è ricaduta sul Python,[9] poiché è il linguaggio principa-

le usato per i programmi che si interfacciano direttamente coi ricercatori, mentre il C++[10] è adottato per i firmware dei dispositivi e i loro driver.

Entrambi i software sono stati realizzati seguendo una struttura modulare: sono composti da un modulo "*widget*", che si occupa dell'interazione col ricercatore e gestisce l'interfaccia grafica, collegato a un modulo "*device*", che dialoga col dispositivo a cui ci si vuole collegare, permette di configurarlo e di acquisirne i dati. Questa separazione ha permesso, ad esempio, di sviluppare velocemente dei programmi per eseguire dei test in automatico: combinando *widget* diversi, è stato possibile acquisire e salvare dati per configurazioni diverse del trasmettitore senza alcun intervento umano, tipicamente un processo laborioso e che richiede molto tempo se svolto a mano.

2.2.1 *Alice Controller*

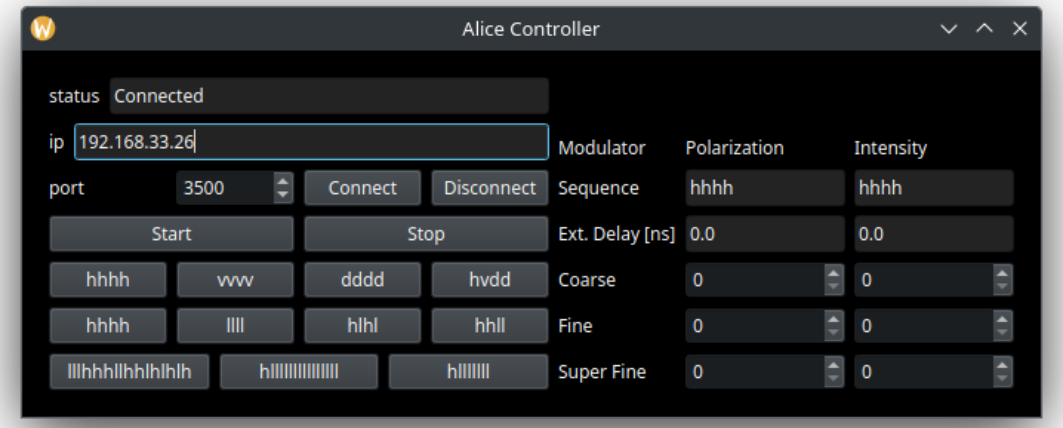


Figura 2.4: Interfaccia grafica di *Alice Controller* all'inizio del progetto.

Alice Controller è stato sviluppato in Python, versione 3.10, e l'interfaccia grafica è stata realizzata con *PyQt 6*.[11] Il *layout* originale è mostrato in figura 2.4: inizialmente si potevano solo inviare chiavi predeterminate, indicate nella pulsantiera nella metà di sinistra, e impostare i ritardi dei segnali di polarizzazione e intensità, tramite gli *spinbox* nella metà di destra. Le modifiche apportate durante il corso del progetto sono risultate in un arricchimento dell'interfaccia grafica e sono descritte di seguito. In figura 2.5 viene mostrato il suo diagramma delle classi: il modulo "*widget*" è omonimo, mentre il modulo "*device*" è chiamato `Zedboard`, dal nome della scheda che integra il SoC e il laser. Anche quest'ultimo modulo era presente all'inizio del progetto, ed è stato aggiornato ove necessario.

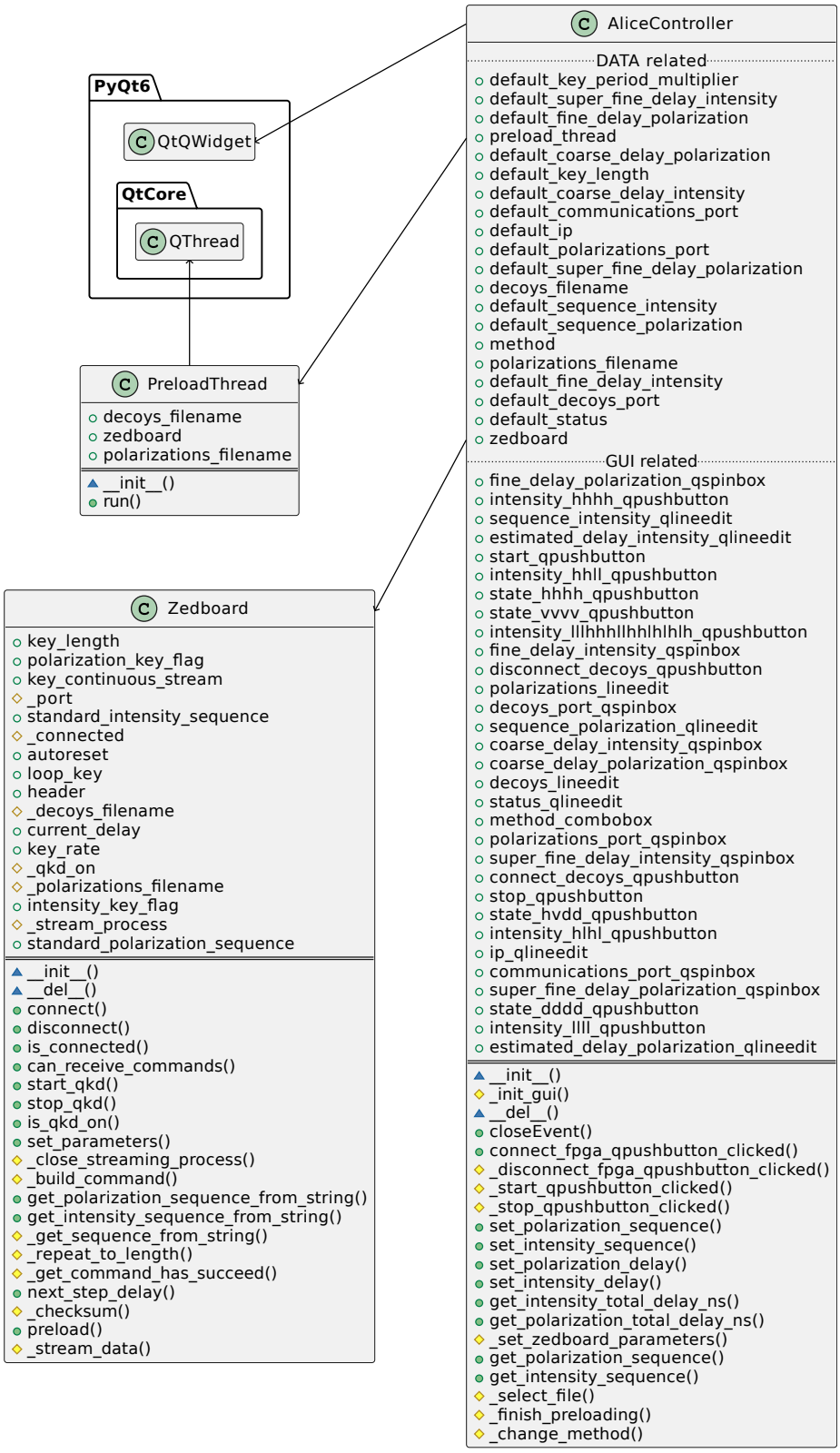


Figura 2.5: Diagramma delle classi di *Alice Controller*.

Quando viene eseguito, *Alice Controller* crea una `QMainWindow` il cui unico widget è `AliceController`. In fase di inizializzazione, vengono definiti i valori di default dei parametri che potranno essere modificati durante l'esecuzione del programma, viene inizializzato `Zedboard` e viene preparato il *layout* dell'interfaccia grafica. Uno di questi parametri è `default_key_length`, ovvero la lunghezza in byte di una chiave da inviare: in modalità *streaming*, il trasmettitore imposta questo valore al massimo, 375 MiB, pari alla metà della sua BRAM, per poter memorizzare e processare contemporaneamente due chiavi in buffer separati.

La comunicazione con Alice avviene via TCP/IP attraverso un collegamento Ethernet, e i tre canali di comunicazione sono dati dai *port* esposti da Alice. Generalmente sono il *port* 3500 per i comandi, 3600 per i "dati di polarizzazione" e 3700 per i "dati d'intensità". Poiché è comunque possibile cambiarli, si è scelto di aggiungere all'interfaccia grafica altri due `QSpinBox`, per entrambi i canali dedicati ai dati, i cui valori di partenza sono i *port* appena discussi. Quando viene premuto il pulsante "Connect", `AliceController` esegue `connect_fpga_qpushbutton_clicked()`: a sua volta esegue `Zedboard.connect()`, che si collega all'indirizzo IP specificato nel campo "IP address", a ciascuno dei *port* selezionati. Se il collegamento è andato a buon fine, `AliceController` esegue `_set_zedboard_parameters()`, configura il trasmettitore coi valori di default menzionati in precedenza, lo imposta in modalità *fixed* e il messaggio di stato viene aggiornato in "Connected", altrimenti viene solo aggiornato in "Error connection".

Se si vuole attivare la modalità *streaming*, è necessario selezionare i file delle due chiavi richieste. Per questo sono stati introdotti due campi di testo, dei `QLineEdit` indicati da "Polariz. file" e "Decoys file", in cui è possibile inserire i loro percorsi, indifferentemente relativi o assoluti. Per comodità, a fianco di ciascun campo di testo, è presente un pulsante "...": se premuto, viene eseguita `AliceController._select_file()`, che apre un selettore di file in una nuova finestra. Se questa finestra non viene chiusa prematuramente e viene selezionato correttamente un file, il campo di testo corrispondente viene aggiornato con il suo percorso assoluto.

Per poter selezionare il metodo con cui inviare la chiave, è stato aggiunto un menù a discesa che presenta due voci, "Fixed" e "Streaming", inizialmente impostato su "Fixed". Ogni volta in cui viene selezionata una voce, `AliceController` esegue `_change_method()`, che prepara il trasmettitore nella modalità corrispondente quando la voce selezionata è diversa da quella attiva.

Quando viene selezionata "Streaming", si entra nella fase di precaricamento, indicata in figura 2.3: vengono prelevati i percorsi dei file delle chiavi dai campi di testo, viene creato un `PreloadThread` e il messaggio di stato viene fissato in "Preloading..." finché l'operazione non è terminata. L'esecuzione di questa fase dipende dalla velocità del collegamento tra il PC e il trasmettitore, perciò si è scelto di eseguirla in un thread dedicato, in modo da non bloccare l'aggiornamento dell'interfaccia grafica per un periodo anche lungo. Inizialmente, il PC imposta il parametro booleano di Alice `key_continuous_stream` a `1`, in modo da abilitare l'invio

continuo dei dati da parte del trasmettitore. È qui che viene impostato al massimo il valore di `default_key_length`, a 375 MiB. Alice viene poi resettata, e riceve la prima parte di entrambe le chiavi. Il procedimento è analogo sia per la chiave delle polarizzazioni che per la chiave delle intensità, le uniche differenze consistono in un cambio di nomi e del canale dei dati, perciò nel seguito sarà descritto solo il primo caso. Il PC imposta `polarization_key_flag` a `1`, indicando ad Alice che dovrà leggere circolarmente il buffer preposto ai "dati di polarizzazione", invece di leggere una volta sola `default_key_length` B. Poi invia il comando `SET_STATES_KEY`: in questo modo Alice si prepara a ricevere i primi 187,5 MiB dei "dati di polarizzazione", pari a metà della lunghezza del buffer dedicato alla chiave, e quando è pronta invia al PC il messaggio `READY_FOR_STATES_KEY`. Il PC allora legge i primi 187,5 MiB dal file della chiave per la polarizzazione e li invia al trasmettitore. Se la chiave è troppo corta, viene ripetutamente concatenata a sé stessa fino ad essere lunga quanto richiesto. I dati vengono inviati nel canale dedicato, e al termine il trasmettitore invia il messaggio `STATES_KEY_TRANSFER_DONE`, insieme a una *checksum* dei dati ricevuti. Il PC controlla che la *checksum* dei dati inviati sia la medesima, nel qual caso il trasferimento è andato a buon fine. Questo `QThread` permette di eseguire una funzione al termine della sua esecuzione: in questo caso viene eseguita `_finish_preloading()` di `AliceController`, che in assenza di errori aggiorna il messaggio di stato in "Preloading done".

Infine, dopo aver eventualmente ritardato i segnali della polarizzazione e dell'intensità, si può attivare la QKD premendo il pulsante "Start". `AliceController` allora esegue `_start_qpushbutton_clicked()`, che a sua volta esegue `Zedboard.start_qkd()`. Il PC invia ad Alice il comando per avviare la QKD, e se è attiva la modalità *streaming*, viene lanciato il processo che invia di continuo le chiavi ad Alice. Se l'attivazione è andata a buon fine, il messaggio di stato viene impostato in "QKD on", altrimenti in "Error QKD on". Alice, nel mentre, consuma la parte iniziale delle chiavi che ha precaricato. Si è scelto di inviare le chiavi in un thread separato per lo stesso motivo della fase di precaricamento: per non appesantire il thread che gestisce l'interfaccia grafica. In questo caso si può usare un processo secondario perché non è necessario modificare lo stato di Alice, ma solo riceverne le richieste e inviarle nuovi dati. Il processo esegue `Zedboard._stream_data()`, che riprende a leggere i file delle chiavi da dove aveva terminato in fase di precaricamento, e ogni volta che Alice richiede una nuova parte di una chiave, legge 18,75 MiB dal file corrispondente e li invia. Se nel mentre viene raggiunta la fine di un file, i byte rimanenti vengono presi dall'inizio del file e la sua lettura ricomincia da capo.

Il formato delle chiavi è il seguente: l'alfabeto della chiave della polarizzazione è composto da tre simboli, `d`, `h`, `v`, che corrispondono alla polarizzazione rispettivamente diagonale, orizzontale e verticale; l'alfabeto della chiave dell'intensità è composto da due simboli, `l`, `h`, che corrispondono all'intensità del fascio bassa e alta. Ogni chiave è una sequenza di questi simboli, che devono essere codificati ciascuno con due bit, rispettivamente 01, 10, 11 e 01, 10. Alice

legge ogni byte come una sequenza di quattro simboli in *little endian*, ovvero la sequenza di quattro simboli $ABCD$ deve essere codificata in un byte nell'ordine inverso, $DCBA$.

Se si vuole ritardare un segnale rispetto all'impulso del laser, nella metà di destra dell'interfaccia grafica sono presenti degli *spinbox* dedicati. Sono disponibili tre livelli di granularità: "*coarse*", a intervalli di 5 ns; "*fine*", tra 200 ps e 500 ps; "*super fine*", a intervalli di 70 ps.

Da questo momento è possibile analizzare la sessione di QKD con *Bob Analyzer*. Quando la si vuole terminare, si può premere il pulsante "Stop" o terminare direttamente il programma. Nel caso in cui sia attiva la modalità *streaming*, verrà ucciso anche il processo secondario.

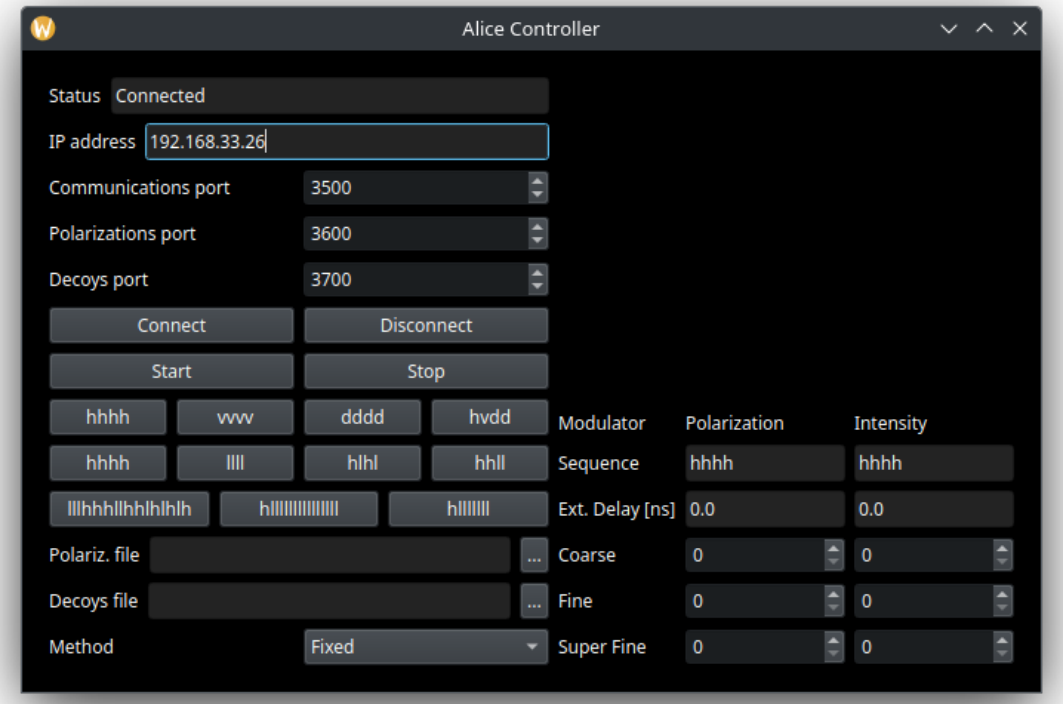


Figura 2.6: Interfaccia grafica di *Alice Controller* al termine del progetto.

2.2.2 *Bob Analyzer*

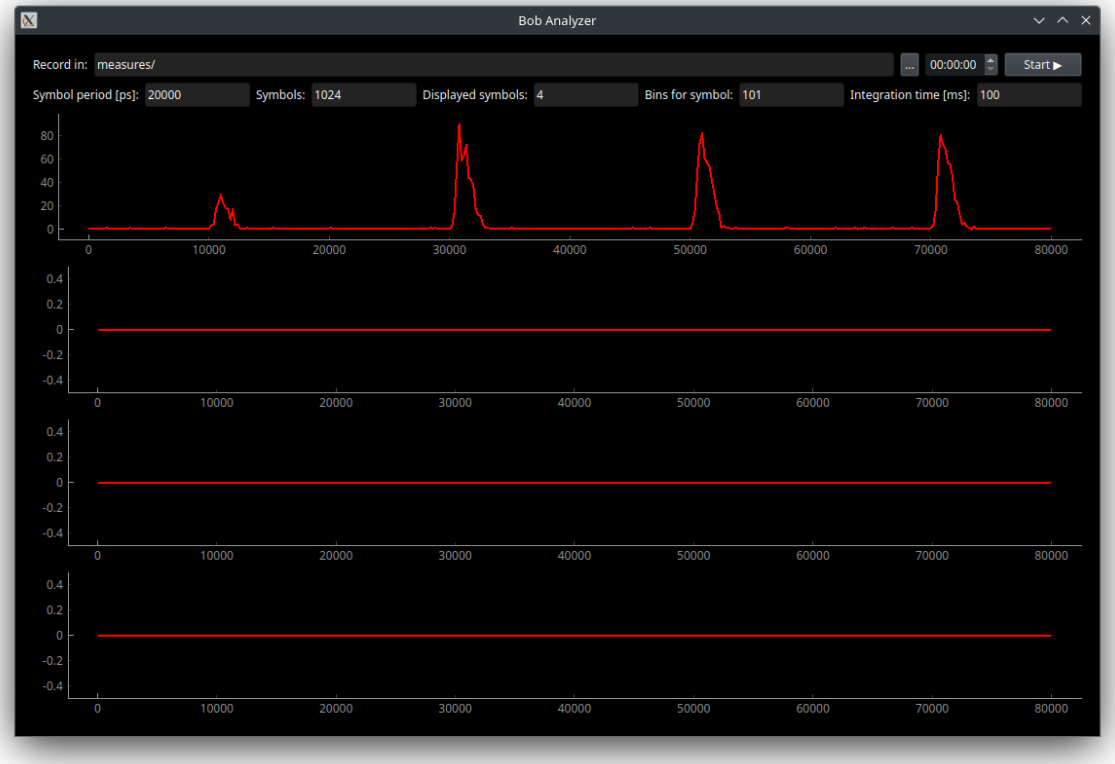


Figura 2.7: Interfaccia grafica di *Bob Analyzer*.

Anche *Bob Analyzer* è stato sviluppato in Python, versione 3.10, e l'interfaccia grafica è stata realizzata con *PyQt 6*, mentre i grafici degli istogrammi sono stati realizzati con *PyQtGraph*.[12] In figura 2.8 è presente il suo diagramma delle classi: il modulo "*widget*" è omonimo, mentre il modulo "*device*" è chiamato `QuTAG`, fornito insieme all'omonimo ricevitore.

L'avvio di *Bob Analyzer* è simile a quello di *Alice Controller*. Il suo widget, `BobAnalyzer`, prepara il *layout* dell'interfaccia grafica, visibile in figura 2.7, in cui vengono mostrati i primi quattro simboli `lhhh` di una chiave trasmessa durante una sessione di QKD. A differenza di `AliceController`, si collega al ricevitore via USB durante l'`__init__()`, e attiva due `QTimer` periodici: `read_timer`, di periodo T_{read}, e `plot_timer`, di periodo T_{int} (`integration_time_ms`), misurati entrambi in millisecondi. Il primo serve per acquisire i dati dal ricevitore, l'altro per calcolare e mostrare sullo schermo gli istogrammi dei dati acquisiti durante l'ultimo T_{int}.

Dopo ogni T_{read}, `read_timer` esegue `read_data()`: tramite `QuTAG.get_timestamps_fullbuffer()` vengono acquisiti gli array dei canali e dei *time tag* immagazzinati nel ricevitore da tutti i suoi canali attivi. Per rendere l'operazione

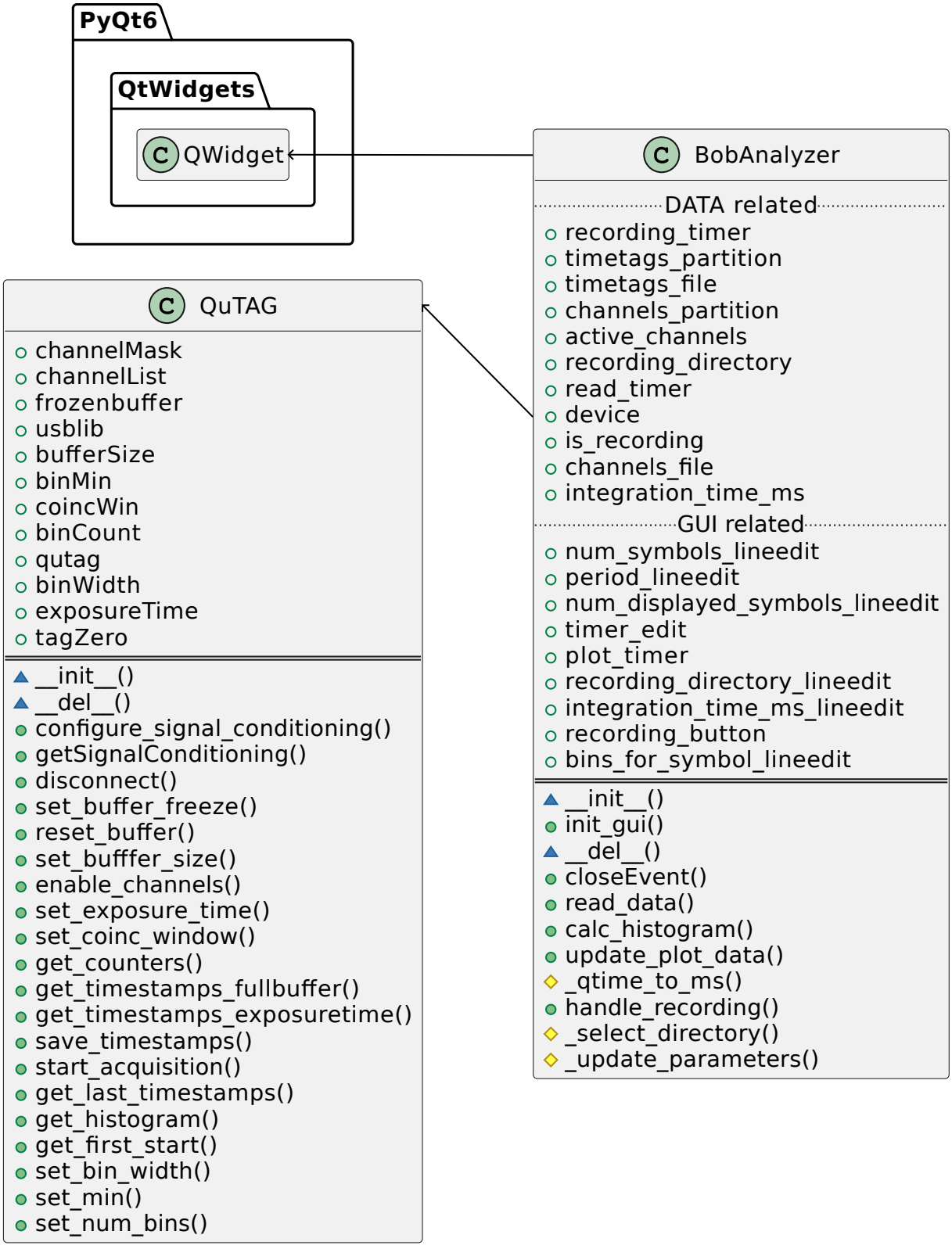


Figura 2.8: Diagramma delle classi di *Bob Analyzer*.

più veloce, non vengono concatenati agli array già acquisiti, ma vengono inseriti ciascuno in una rispettiva lista, per essere concatenati in seguito. Poiché gli array restituiti da `QuTAG.get_timestamps_fullbuffer()` sono circolari, se necessario i dati vengono riordinati in modo che l'array dei *time tag* sia monotono crescente, e di conseguenza viene riordinato l'array dei canali.

Passato un T_{int}, `plot_timer` esegue `update_plot_data()`: gli array contenuti nelle liste di canali e *time tag* vengono concatenati, e le liste svuotate. Poi si procede a calcolare gli istogrammi. Per ogni canale attivo, vengono prelevati i *time tag* relativi al canale corrente, poi viene eseguito `calc_histogram()`: viene calcolato il resto della divisione tra ciascun *time tag* e il periodo della sequenza di simboli che si sta ricevendo T_{seq}, dato dal prodotto tra il numero di simboli N_{seq} e il periodo di ciascun simbolo T_s. I valori risultanti sono contenuti nell'intervallo $[0, T_{\text{seq}})$, mentre gli istanti relativi all'i-esimo simbolo sono contenuti nell'intervallo $[iT_s, (i+1)T_s)$. Viene poi calcolato l'istogramma di questi valori, con una quantità di intervalli pari al numero di intervalli per simbolo N_{bin} per il numero di simboli N_{seq}. Infine, vengono aggiornati i dati del grafico del canale corrente e *PyQtGraph* si occupa di ridisegnarlo. In particolare, viene mostrato l'istogramma dei primi N_{hist} simboli, ovvero dei primi $N_{\text{bin}}N_{\text{hist}}$ intervalli.

Per quando è necessario salvare i dati ricevuti, l'interfaccia grafica espone un campo di testo in cui inserire il percorso della cartella in cui si vuole salvare i file (`recording_directory`). Per convenienza, è presente un pulsante che apre un selettore di file in una nuova finestra. Nel caso venga selezionata correttamente una cartella, il campo di testo viene aggiornato con il suo percorso assoluto. Se il campo di testo è vuoto, la `recording_directory` sarà la cartella corrente di lavoro. Quando viene premuto il pulsante "Start", `BobAnalyzer` esegue `handle_recording()`: viene attivato il salvataggio dei dati, e vengono creati e aperti i due file in cui salvare rispettivamente i canali e i *time tag*. Il loro nome è determinato come segue: viene calcolato l'istante di tempo in cui viene premuto il pulsante e viene convertito in stringa (`now`) secondo il formato `%Y-%m-%d_%H-%M-%S`, allora i nomi sono `<recording_directory>/<now>_{channels,timetags}.bin`. Poi il testo del pulsante viene aggiornato in "Stop". Se il pulsante viene premuto di nuovo, il salvataggio dei dati viene terminato, i file aperti vengono chiusi, e il testo del pulsante viene ripristinato in "Start". Infine, è possibile impostare la durata del `recording_timer` che, dopo aver avviato il salvataggio dei dati, lo terminerà in automatico allo scadere del tempo. Quando il timer è attivo, è comunque possibile terminare il salvataggio dei dati manualmente premendo il pulsante "Stop", disabilitando così anche il timer. Le misurazioni vengono salvate in `read_data()`, subito dopo aver riordinato gli array acquisiti. Ciascun array viene scritto nel rispettivo file nello stesso formato in cui è stato ottenuto: un blocco contiguo di numeri interi a 8 bit per l'array dei canali, un blocco contiguo di numeri interi a 64 bit per l'array dei *time tag*. Vengono evitate elaborazioni intermedie dei dati, a differenza di Daisy, che li converte nel formato CSV prima di salvarli. In

questa maniera si risparmiano tempo, senza impattare significativamente sulle prestazioni del programma, e spazio sul disco. Nel corso del progetto sono stati provati altri metodi per salvare i dati acquisiti, ma si sono rivelati inadeguati a causa delle elaborazioni intermedie, che portavano i file ad avere dimensioni eccessive.

Infine, l'interfaccia grafica espone un campo di testo (`QLineEdit`) per ciascuno dei parametri modificabili, in cui è possibile inserire un valore intero. Viene chiaramente indicato a quale parametro è associato il campo di testo, e la correttezza del valore inserito viene verificata da un `QIntValidator`. Quando viene premuto il tasto ⏎, viene eseguita `_update_parameters()`: se il testo inserito è un numero intero, il valore del parametro corrispondente viene aggiornato, altrimenti non accade nulla. In particolare, se il parametro modificato è T_{int}, il periodo del `plot_timer` viene aggiornato per riflettere la suddetta modifica.

Capitolo 3

Collaudo

3.1 Apparato sperimentale

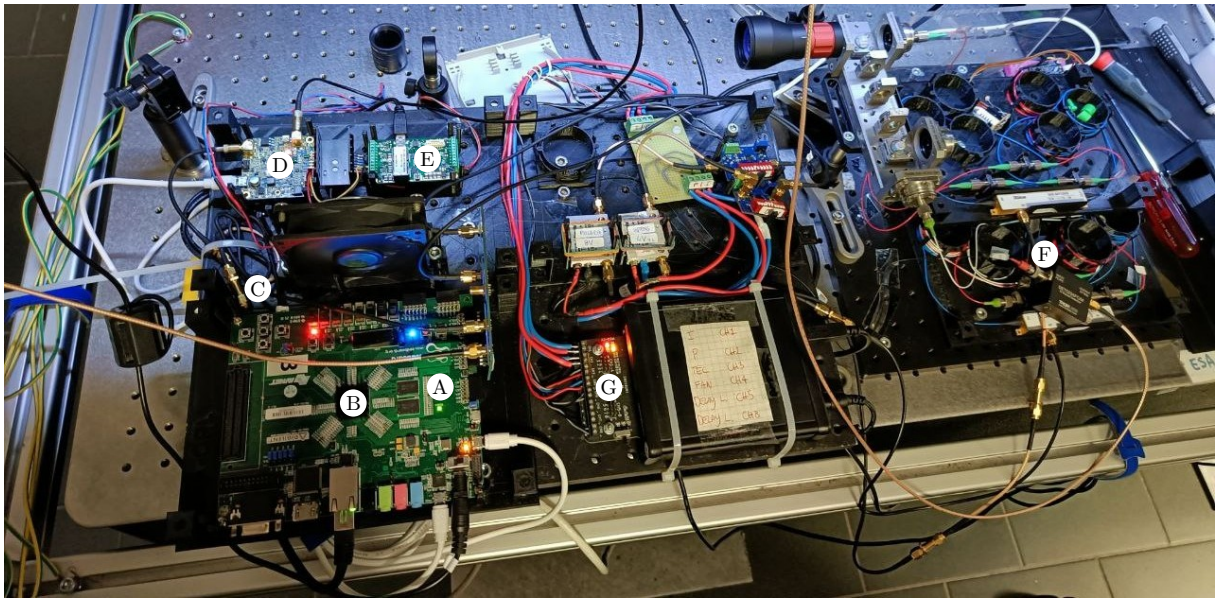


Figura 3.1: *Setup* di Alice usato per gli esperimenti.
I componenti sono: (A) la scheda di sviluppo ZedBoard di Avnet,[13] che comprende (B) il SoC Xilinx Zynq-7000 di AMD;[14] al di sotto della scheda, (C) il diodo laser a singola frequenza EYP-DFB-0795 di EAGLEYARD Photonics;[15] (D) il *driver* ed (E) il raffreddatore termoelettrico del laser, (F) il modulatore di intensità e (G) il generatore che alimenta il sistema.
Non visibile in figura è Bob, composto da un *single-photon counting module for near-infrared* (SPCM-NIR) di Excelitas,[16] collegato al convertitore *time-to-digital* quTAG di qutools.[8]

Il sistema di QKD usato nel corso del progetto è visibile nella figura 3.1, ed è basato sullo schema in figura 1.1. Per collaudare i software, è stato disattivato il modulatore della polarizzazione, ed è stato usato solo il modulatore dell'intensità, alternando i cavi uscenti dalla scheda

per modulare l'intensità e la polarizzazione sempre e solo in quest'ultimo. Per questo motivo, i test che hanno coinvolto il segnale di polarizzazione hanno necessitato di una fase di calibrazione (ruotare la lamina del polarizzatore), per poter mappare i tre stati generati in tre livelli di intensità diversi. Ciò non è servito per il segnale di intensità, poiché sono stati sufficienti due livelli.

3.2 Test dei software

Per procedere all'analisi dei dati, descritta nella sezione 3.3, è stato necessario verificare il corretto funzionamento di *Alice Controller* e *Bob Analyzer*. Per farlo sono stati eseguiti dei test descritti di seguito.

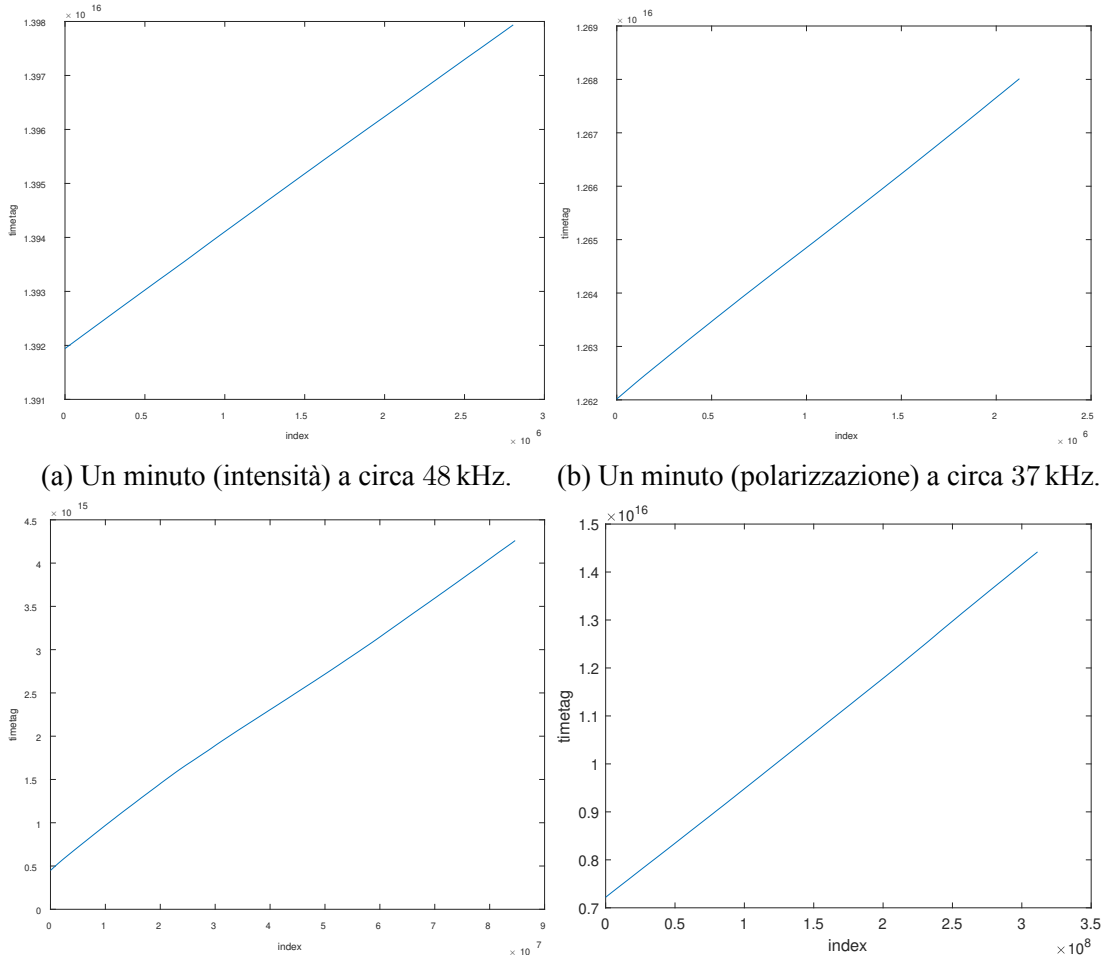
3.2.1 Conformità delle chiavi generate al formato richiesto

Le chiavi inviate durante il corso di questo progetto sono state generate da un programma sviluppato appositamente. Il suo funzionamento è il seguente: viene definita ciascuna chiave come una stringa dei caratteri ammissibili, inizialmente lunga a piacere; poiché in ciascun byte devono essere codificati quattro simboli, se il numero di simboli non è divisibile per 4, viene ripetuta 4 volte. La chiave viene quindi convertita nel formato richiesto, e viene salvata su un file, ripetendola finché la lunghezza del file non è pari a $187{,}5$ MiB, in modo da poter essere inviato interamente durante la fase di precaricamento della modalità *streaming*. Per ciascuna chiave, viene anche generato un file che contiene la lista dei loro simboli, utile durante l'analisi dei dati.

È stato necessario assicurarsi che i file da inviare al trasmettitore rispettassero il formato descritto al termine della sezione 2.2. Si sono svolte delle verifiche sia conducendo un'analisi dell'algoritmo di conversione, sia leggendo le chiavi che Alice ha ricevuto e comunicato al PC attraverso un collegamento seriale, e non sono stati rivelati errori. Una verifica aggiuntiva è data dalla *checksum* delle chiavi ricevute che Alice invia al PC durante la fase di precaricamento, mostrata in figura 2.3.

3.2.2 Assenza di perdite di dati durante il salvataggio

Per poter condurre delle analisi più approfondite sui dati ricevuti, era necessario che *Bob Analyzer* non ne perdesse durante il loro salvataggio. Per questo sono state eseguite delle misurazioni di varia durata, sono stati disegnati i grafici dei *time tag* ottenuti e si è verificato che non presentassero discontinuità.



(a) Un minuto (intensità) a circa 48 kHz.

(b) Un minuto (polarizzazione) a circa 37 kHz.

(c) Un'ora a circa 24 kHz.

(d) Due ore a circa 43 kHz.

Figura 3.2: Grafici dei dati acquisiti durante quattro sessioni di QKD.
I valori delle ascisse corrispondono agli indici del *time tag*, quelli delle ordinate ai *time tag* stessi in picosecondi.

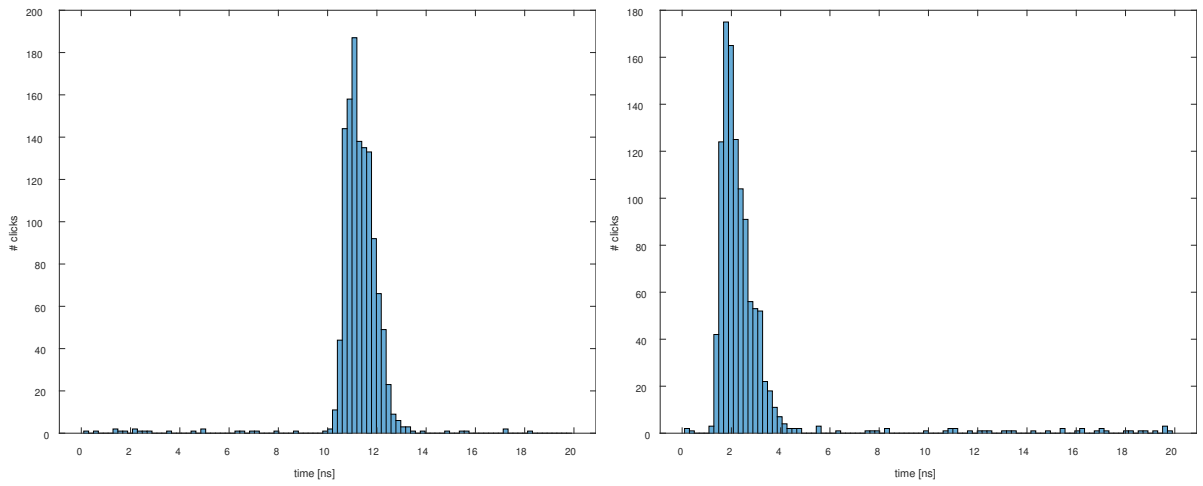
In figura 3.2 sono riportati i grafici di quattro misurazioni eseguite durante delle sessioni di QKD, due da un minuto, una da un'ora e una da due ore. Come è possibile notare, sono assenti discontinuità che facciano pensare a un'eventuale perdita di dati. Inoltre, poiché i grafici sono monotoni crescenti, si evince anche il corretto riordinamento dei buffer circolari acquisiti dal ricevitore, discusso nella sezione 2.2.2.

3.2.3 Individuazione dei simboli trasmessi

Prima di poter verificare la corretta trasmissione di una chiave in modalità *streaming*, si è visto necessario verificare che il trasmettitore inviasse effettivamente dei simboli. Ciò significa che, nel periodo di tempo che il trasmettitore impiegava per inviare un simbolo T_s, il ricevitore doveva registrare più misurazioni ravvicinate, mentre tra due invii consecutivi non doveva

essere registrata alcuna misurazione, a meno del rumore, che comunque doveva influenzare la trasmissione il meno possibile.

Per questo sono state eseguite alcune sessioni di QKD, in cui il ricevitore è stato configurato con un canale attivo e uno di sincronizzazione. Sono state salvate delle misurazioni di un minuto e le si sono poi analizzate con MATLAB. Dopo aver caricato i dati dei canali e dei *time tag*, si sono estratti i *time tag* del canale attivo, è stato calcolato il resto di ciascuno di essi rispetto al periodo della sequenza inviata $T_{\text{seq}} = T_s N_{\text{seq}}$, infine si è calcolato e disegnato l'istogramma dei valori ottenuti. In figura 3.3 sono mostrati gli istogrammi nel periodo del primo simbolo di due sessioni di QKD, con $T_s = 20\,\text{ns}$ ed $N_{\text{bin}} = 101$, in cui nella prima la chiave è stata inviata nel segnale di intensità, nella seconda nel segnale di polarizzazione. Il numero di conteggi è indicato nelle ordinate come "# clicks". Il fatto che i grafici presentino un picco di conteggi significa che il trasmettitore ha inviato un simbolo entro il periodo stesso. I conteggi al di fuori del picco sono dovuti al rumore.



(a) Segnale di intensità.

(b) Segnale di polarizzazione.

Figura 3.3: Istogrammi del periodo del primo simbolo in due sessioni di QKD. La presenza di un picco indica la trasmissione di un simbolo.

3.2.4 Trasmissione di una chiave nel segnale corrispondente

Inoltre, è stato necessario verificare che una chiave, inviata al trasmettitore come "dati di polarizzazione" o "dati di intensità", venisse trasmessa nel segnale corrispondente. Per questo sono state eseguite delle sessioni di QKD in cui è stata inviata una chiave casuale in uno dei due canali, e una chiave di un solo simbolo nell'altro, ad esempio una chiave composta da soli `h`. Dalle analisi condotte con *Bob Analyzer*, si è visto che il trasmettitore si comportava secondo le aspettative.

3.2.5 Trasmissione continua di una chiave

Infine, si è dovuto verificare che il trasmettitore inviasse continuamente una chiave in modalità *streaming*. Dalle analisi dell'algoritmo di `Zedboard._stream_data()` (figura 2.5), si è visto che i file delle chiavi venivano correttamente riletti da capo dopo essere stati inviati interamente, e da alcune sessioni di QKD è stato verificato che la trasmissione continuasse finché non veniva terminata, direttamente premendo il pulsante "Stop" di *Alice Controller* o indirettamente resettando il sistema di QKD.

3.2.6 Dimensione dei file dei dati salvati

Uno dei motivi che hanno portato allo sviluppo di *Bob Analyzer*, ma che esula dal corretto funzionamento del sistema di QKD, riguardava il peso dei file generati durante il salvataggio dei dati. In precedenza, il compito di salvare i dati veniva affidato a Daisy, un software fornito insieme al ricevitore quTAG.[8] Questo software salva i dati ricevuti convertendoli in CSV, uno per riga nel formato `<time tag>;<channel>\n`. La precisione dei *time tag* è al picosecondo rispetto all'accensione del dispositivo, perciò in CSV ciascuno occupa almeno $12\,\mathrm{B}$, mentre i canali sono numerati da 1 a 4, più il canale 104 per la sincronizzazione con un clock esterno, perciò occupano da $1\,\mathrm{B}$ a $3\,\mathrm{B}$ ciascuno. Considerando anche i caratteri `;` e `\n`, ciascuno pesante $1\,\mathrm{B}$, nelle misurazioni effettuate ogni riga dei file generati pesava almeno $18\,\mathrm{B}$. Ad una frequenza di misurazione di circa $50\,\mathrm{kHz}$, venivano generati almeno $3\,\mathrm{GiB}$ di dati ogni ora. *Bob Analyzer*, invece, salva i dati nello stesso formato in cui li acquisisce: un array di numeri interi da $1\,\mathrm{B}$ per i canali e $8\,\mathrm{B}$ per i *time tag*. In questo modo i file generati pesano generalmente la metà, e ciò ha permesso di risparmiare spazio o di effettuare più misurazioni con la stessa capacità di memoria. I dati salvati possono essere elaborati successivamente, se necessario convertendoli in un formato diverso. Per gli scopi del progetto ciò non è servito, poiché, effettuando le analisi in MATLAB, è stato possibile leggere i file direttamente come array di numeri interi.

3.3 Analisi dei dati

Lo scopo di questa sezione riguarda la verifica dell'obiettivo principale del progetto, ovvero l'invio corretto e continuo di una chiave di 1024 simboli in modalità *streaming*.

Le analisi sono volte a verificare la correttezza della trasmissione della chiave: conoscendo la sequenza di simboli trasmessa S, si è cercato di ricostruirla a partire dai dati ricevuti. Per farlo, è stato necessario individuare il simbolo corretto da associare a ciascun picco degli istogrammi, un esempio dei quali è riportato in figura 3.3. Dalla sequenza S' così ottenuta, la si è scorsa circolarmente fino ad ottenere la correlazione massima con la chiave trasmessa.

Nel seguito sono presentate le analisi condotte su due sessioni di QKD dalla durata di un minuto. In ciascuna è stata generata una chiave di 1024 simboli, in maniera pseudocasuale, ed è stata inviata ad Alice rispettivamente nel canale dei "dati di intensità" e in quello dei "dati di polarizzazione". Poiché la modalità *streaming* richiede di inviare due chiavi al trasmettitore, la chiave inviata nell'altro canale è stata `h` in entrambe le sessioni. Il ricevitore è stato configurato con un canale attivo e un canale di sincronizzazione, e il programma usato per effettuare le analisi è stato sviluppato in MATLAB.

3.3.1 Segnale di intensità

Nel corso di questa sessione, è stata trasmessa una chiave di lunghezza $N_{\text{seq}} = 1024$ simboli nel segnale di intensità, mentre nel segnale di polarizzazione è stata trasmessa la chiave `h`. Il periodo di ciascun simbolo è stato fissato a $T_s = 20\,\text{ns}$.

Inizialmente, sono stati caricati i dati acquisiti con *Bob Analyzer* nell'arco di un minuto. Sono stati poi prelevati i *time tag* acquisiti dall'unico canale attivo, e si è calcolato il resto della divisione tra ciascun *time tag* e il periodo della sequenza $T_{\text{seq}} = T_s N_{\text{seq}}$, in modo da ottenere valori compresi in $[0, T_{\text{seq}})$. Si è poi calcolato l'istogramma, il cui numero di intervalli per simbolo è stato fissato a $N_{\text{bin}} = 101$, e dunque il numero di intervalli totali è dato da $N_{\text{bin}} N_{\text{seq}}$.

Da qui è stato necessario calcolare l'intervallo temporale, durante ogni T_s, che presentava il maggior numero di conteggi, il picco, in cui cioè era più probabile che fosse stato trasmesso un simbolo. In questo modo si è esclusa la maggior parte del contributo del rumore tra la trasmissione di un simbolo e il suo successivo. Per gli scopi di questo progetto, è stato sufficiente trovare gli estremi manualmente: si è disegnato l'istogramma del primo simbolo, visibile in figura 3.3a, e si è determinato con approssimazione sufficiente che il simbolo fosse stato trasmesso tra $5\,\text{ns}$ e $15\,\text{ns}$. La periodicità T_s dell'impulso del laser ha garantito che anche tutti gli altri picchi si sarebbero trovati entro quell'intervallo.

Per ciascun T_s, sono stati integrati i conteggi dell'istogramma entro l'intervallo appena determinato, e si è così ottenuta una sequenza di N_{seq} valori, a cui si sarebbe dovuto associare uno dei simboli ammissibili per il segnale di intensità. Poiché i simboli possibili sono due, corrispondenti all'intensità alta e bassa del segnale, per farlo è bastato controllare se un valore fosse minore della media di tutti i valori (1) o maggiore (0). L'istogramma di questi valori, con un numero di intervalli pari a $N_{\text{bin}} = 100$, è mostrato in figura 3.4. La media delle intensità è $m \approx 2717$, normalizzata rispetto al massimo delle intensità è circa $0{,}614$ ed è indicata nella figura con una linea verticale. La varianza delle intensità è $\sigma^2 \approx 1{,}777 \times 10^6$. L'assegnazione dei simboli ai conteggi è stata immediata grazie alla netta separazione tra i valori, come si può vedere dalla figura.

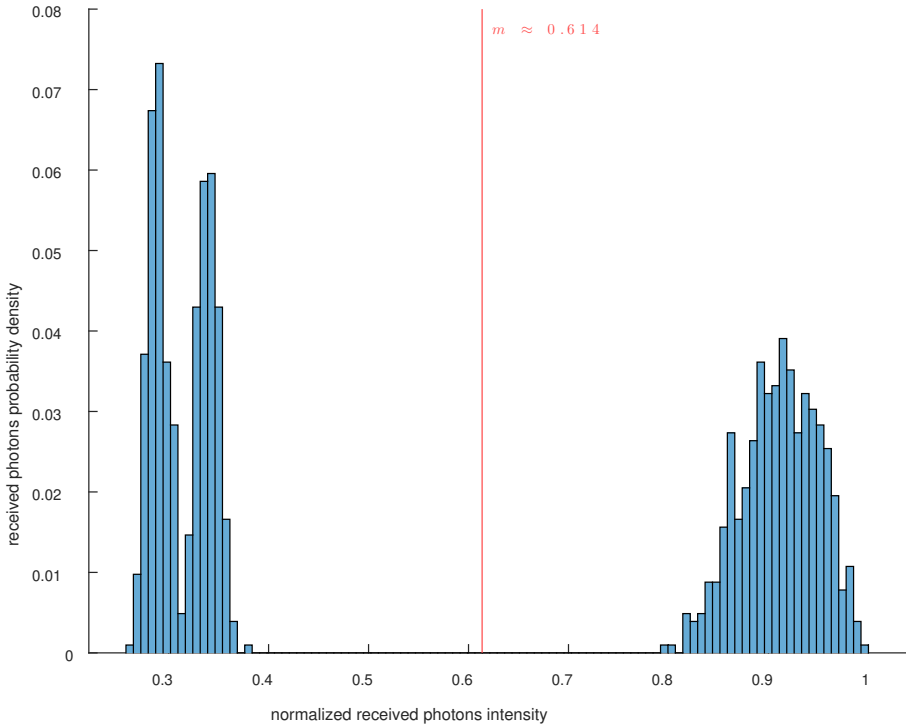


Figura 3.4: Densità di probabilità dei simboli in arrivo per il segnale di intensità.

Così si è ottenuta una sequenza S' di N_{seq} simboli, che sarebbe dovuta corrispondere alla chiave trasmessa S. Tuttavia, non necessariamente si ha $S'_i = S_i$: poiché i dati vengono acquisiti a partire da un istante di tempo arbitrario, è possibile che il primo simbolo ricevuto corrisponda al j-esimo simbolo trasmesso, e così $S'_i = S_{i+j}$, con $i + j$ modulo N_{seq}. Perciò si è visto necessario calcolare la correlazione incrociata tra le due sequenze, ovvero contare per quanti i vale $S_i = S'_i$, scorrendo S' circolarmente N_{seq} volte. Il risultato della correlazione incrociata è mostrato in figura 3.5: come si nota, le due sequenze non sono generalmente perfettamente correlate, tranne in uno degli scorrimenti, in cui la correlazione è massima ed è pari ad N_{seq}.

Ciò significa che la sequenza ottenuta è identica a quella trasmessa, a meno di uno scorrimento circolare. Dunque, per quanto concerne il segnale di intensità, l'obiettivo del progetto è stato raggiunto.

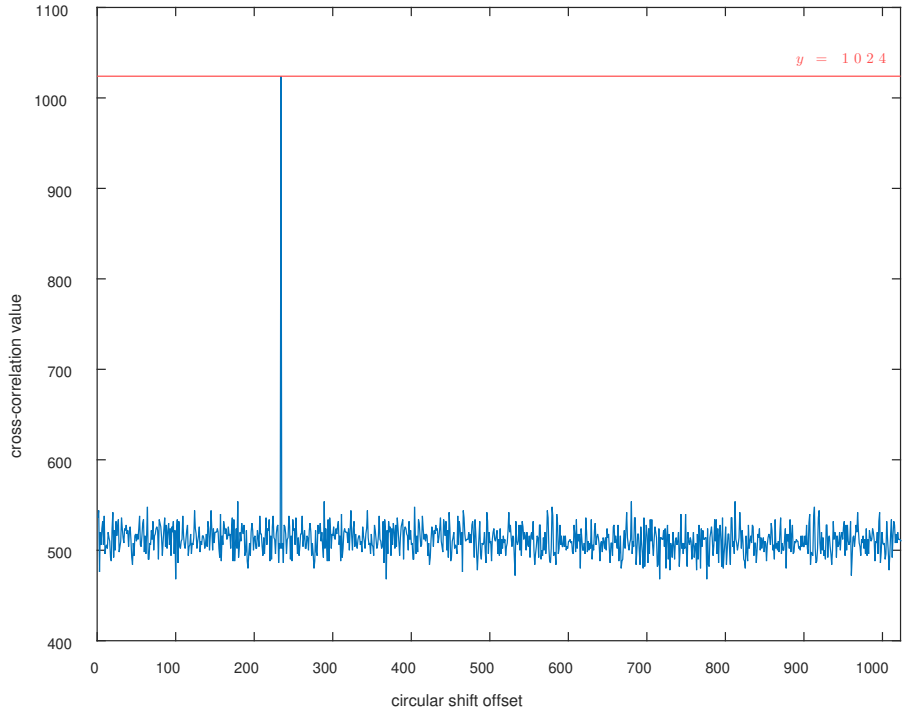


Figura 3.5: Correlazione incrociata tra S ed S' (segnale di intensità). Il valore massimo è 1024 su 1024.

3.3.2 Segnale di polarizzazione

Nel corso di questa sessione è stata trasmessa una chiave di lunghezza $N_{\text{seq}} = 1024$ simboli nel segnale di polarizzazione, mentre nel segnale di intensità è stata trasmessa la chiave `h`. Il periodo di ciascun simbolo è stato fissato a $T_s = 20\,\text{ns}$ e gli istogrammi delle figure 3.3b e 3.6 hanno N_{bin} rispettivamente pari a 101 e 100. Gli estremi dell'intervallo in cui è visibile il picco sono stati determinati in $0\,\text{ns}$ e $5\,\text{ns}$. L'analisi è stata condotta in maniera identica rispetto a quanto discusso in 3.3.1.

Come si può notare dalla figura 3.6, le densità delle tre intensità sono ben separate, e ciò ha permesso una veloce associazione ai simboli 2, 0 e 1, rispettivamente da sinistra a destra. Per quanto concerne questo progetto, i due separatori sono stati determinati manualmente in $x_1 = 1000$ e $x_2 = 1800$, normalizzati rispetto al massimo delle intensità circa in $0{,}405$ e $0{,}729$, e sono segnati in figura con due linee verticali. La varianza è $\sigma^2 \approx 4{,}215 \times 10^5$.

La correlazione incrociata, in figura 3.7, ha visto un massimo in 1024 su una sequenza di lunghezza 1024. Anche in questo caso l'obiettivo del progetto è stato raggiunto. Tuttavia, questo risultato è stato ottenuto analizzando all'incirca i primi $45\,\text{s}$ dei $60\,\text{s}$ totali: da poco oltre i $55\,\text{s}$, si è presentato un graduale *drift* dei dati, che ha portato alla sovrapposizione delle densità. Al sessantesimo secondo, non è stato possibile ricostruire 6 simboli su 1024, con un errore di circa lo $0{,}59\%$.

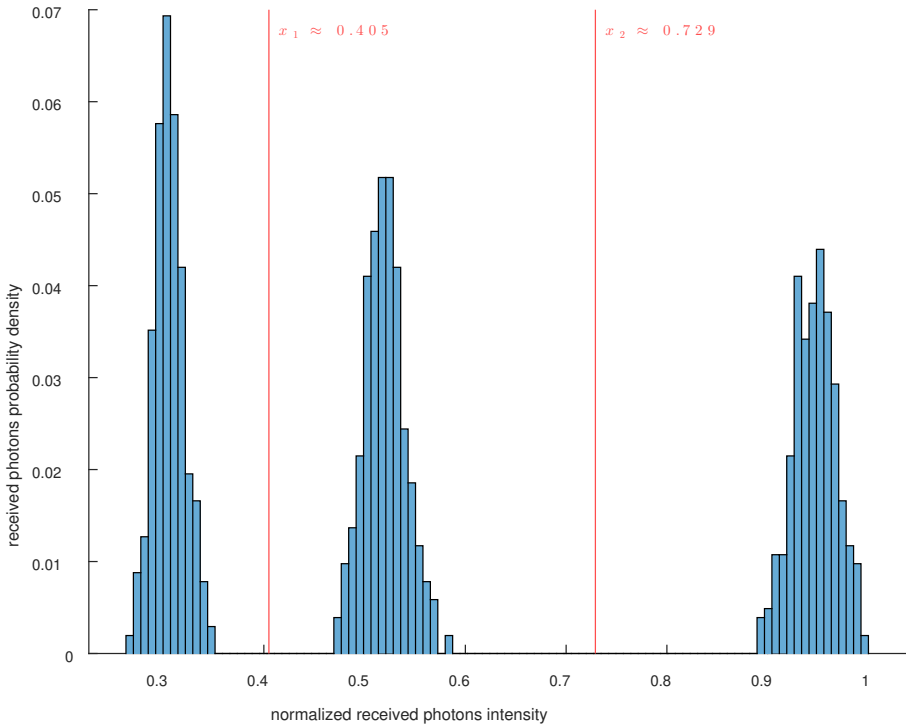


Figura 3.6: Densità di probabilità dei simboli in arrivo per il segnale di polarizzazione.

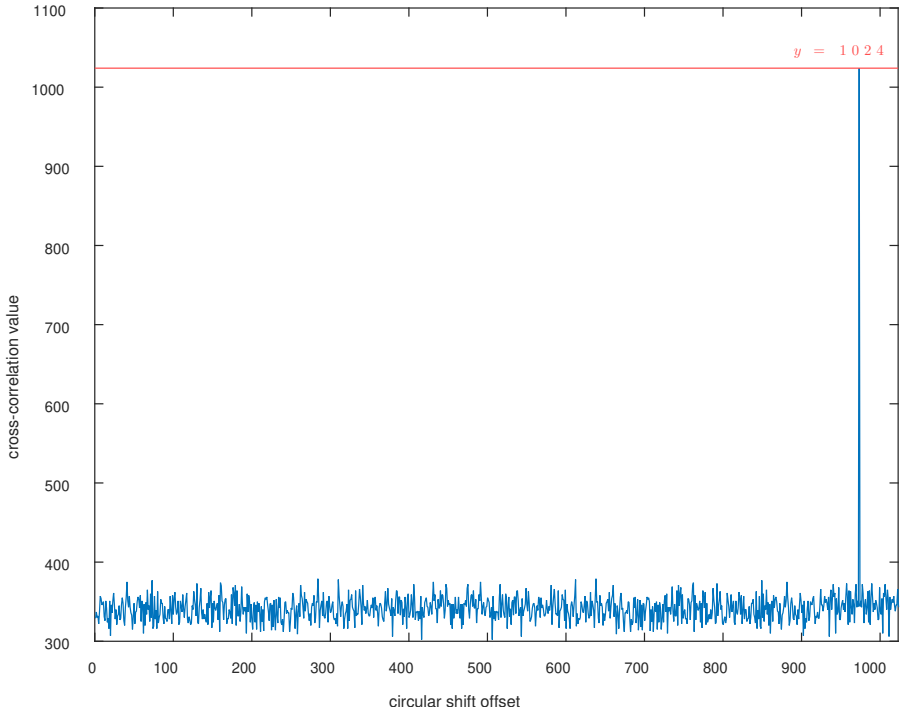


Figura 3.7: Correlazione incrociata tra S ed S' (segnale di polarizzazione). Il valore massimo è 1024 su 1024.

Capitolo 4

Conclusioni

Questo progetto ha avuto come scopo lo sviluppo di software in Python per la gestione di un sistema di *quantum key distribution*.

È stato aggiornato il programma per l'invio della chiave crittografica al trasmettitore, *Alice Controller*. È stata implementata la modalità di trasmissione *streaming*, che permette di inviare continuamente due chiavi di lunghezza arbitraria, mentre in precedenza era possibile inviare solo chiavi prefissate.

Inoltre, è stato sviluppato un programma per visualizzare e salvare i dati ricevuti, *Bob Analyzer*, più efficiente rispetto a quello fornito insieme al ricevitore: se prima il salvataggio dei dati, a circa 50 mila misurazioni al secondo, generava intorno ai $3\,\mathrm{GiB\,h^{-1}}$, col sistema attuale vengono prodotti circa $1{,}5\,\mathrm{GiB\,h^{-1}}$. Questo è possibile perché i dati vengono salvati così come sono ricevuti: eventuali elaborazioni, come la conversione nei formati supportati da MATLAB per le successive analisi, vengono eseguite al termine del salvataggio. Le analisi condotte sul software non hanno mostrato perdite di dati. Per quanto concerne la visione in tempo reale dei dati ricevuti, viene mostrato l'istogramma della sola porzione dei dati desiderata: l'elaborazione è più veloce e il suo carico computazionale non impatta in maniera significativa l'uso del programma. Si possono variare a piacimento alcuni parametri, che ne modificano istantaneamente il funzionamento.

L'obiettivo del progetto di inviare continuamente una chiave di 1024 simboli è stato soddisfatto per entrambi i segnali di intensità e di polarizzazione, ma la presenza di un *drift* per quest'ultimo suggerisce una instabilità dell'apparato di misura.

Risolto questo problema, si potrà adottare questo sistema per implementare dei protocolli di *quantum key distribution*, e realizzare dei prototipi completi e automatici per portare la QKD al di fuori dei laboratori di ricerca. Per la generazione delle chiavi sarebbe auspicabile usare un generatore di numeri casuali quantistico, così da sfruttare il probabilismo intrinseco della meccanica quantistica, per migliorare la qualità delle chiavi trasmesse.

Bibliografia

[1] G. Benenti, G. Casati e G. Strini, *Principles of Quantum Computation and Information*. WORLD SCIENTIFIC, 2004, cap. 1.1, 1.2. doi: 10.1142/5528. eprint: https://www.worldscientific.com/doi/pdf/10.1142/5528. indirizzo: https://www.worldscientific.com/doi/abs/10.1142/5528.

[2] C. H. Bennett e G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," in *Proceedings of the IEEE International Conference on Computers, Systems, and Signal Processing, Bangalore, Dec. 1984*, 1984, pp. 175–179.

[3] Università degli Studi di Padova. "QuantumFuture." (2023), indirizzo: https://quantumfuture.dei.unipd.it/.

[4] F. Berra, C. Agnesi, A. Stanco et al., *Modular source for near-infrared quantum communication*, 2023. arXiv: 2301.12882 [quant-ph].

[5] M. Avesani, C. Agnesi, A. Stanco, G. Vallone e P. Villoresi, "Stable, low-error, and calibration-free polarization encoder for free-space quantum communication," *Optics Letters*, vol. 45, n. 17, p. 4706, 2020. doi: 10.1364/ol.396412. indirizzo: https://doi.org/10.1364%2Fol.396412.

[6] W.-Y. Hwang, "Quantum Key Distribution with High Loss: Toward Global Secure Communication," *Physical Review Letters*, vol. 91, n. 5, 2003. doi: 10.1103/physrevlett.91.057901. indirizzo: https://doi.org/10.1103%2Fphysrevlett.91.057901.

[7] B. Qi, L. Qian e H.-K. Lo, "State of the art QKD technologies," in *A brief introduction of quantum cryptography for engineers*. 2010, cap. 3, pp. 15–25. arXiv: 1002.1237 [quant-ph].

[8] qutools GmbH. "quTAG." (2023), indirizzo: https://qutools.com/qutag/.

[9] Python Software Foundation. "Welcome to Python.org." (2023), indirizzo: https://www.python.org/.

[10] ISO/IEC JTC 1/SC 22/WG 21. "Standard C++." (2023), indirizzo: https://isocpp.org/.

[11] Riverbank Computing Limited, The Qt Company. “Reference Guide — PyQt Documentation v6.” (2023), indirizzo: https://www.riverbankcomputing.com/static/Docs/PyQt6/.

[12] PyQtGraph. “PyQtGraph - Scientific Graphics and GUI Library for Python.” (2023), indirizzo: https://www.pyqtgraph.org/.

[13] Avnet, Inc. “ZedBoard.” (2023), indirizzo: https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/zedboard/.

[14] Advanced Micro Devices, Inc. “Zynq 7000 SoC.” (2023), indirizzo: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

[15] EAGLEYARD Photonics GmbH. “EYP-DFB-0795-00040-1500-BFW11-0005.” (2023), indirizzo: https://www.toptica-eagleyard.com/ey-product/eyp-dfb-0795-00040-1500-bfw11-0005/.

[16] Excelitas Technologies Corp. “SPCM-NIR.” (2023), indirizzo: https://www.excelitas.com/product/spcm-nir.