

Quantum Key Distribution GUI

A Real-Time Data Visualization Tool

Manikant Kumar (PIP-42)

Under the guidance of

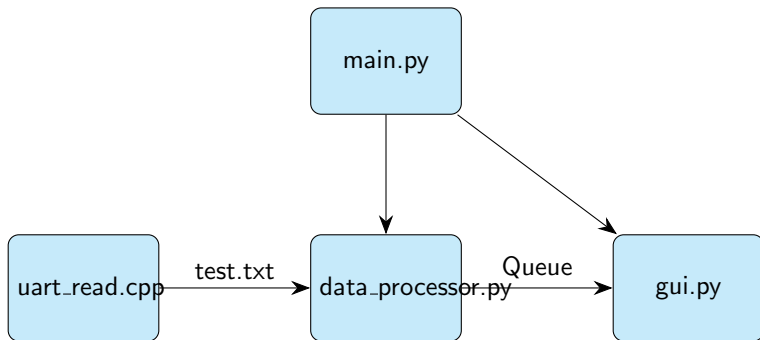
Dr. XYZ

July 28, 2025

Project Overview

- **Purpose:** Visualize Quantum Key Distribution (QKD) metrics in real-time.
- **Features:**
 - Tabbed GUI with histograms and line plots
 - Supports file and console (UART) data input
 - Real-time visualization of QBER, throughput, visibility, keys
 - Session management and mode switching
- **Technologies:**
 - Python: PyQt6, pyqtgraph, pyserial, NumPy
 - C++: Windows API, C standard library

System Architecture



- `main.py` initializes application
- `uart_read.cpp` reads serial data to `test.txt`
- `data_processor.py` parses data and queues it
- `gui.py` visualizes data in real-time

File Analysis: main.py

- **Purpose:** Application entry point
- **Key Functions:**
 - Initializes mode, DataProcessor, and MainWindow
 - Sets up PyQt6 application loop
- **Code Snippet:**

```
1  def main():
2      mode = "file"
3      if len(sys.argv) > 1 and sys.argv[1] in ["console", "file"]:
4          mode = sys.argv[1]
5      data_queue = Queue()
6      processor = DataProcessor(data_queue, mode=mode, input_string="default_input")
7      app = QApplication(sys.argv)
8      window = MainWindow(data_queue, processor)
9      window.show()
10     sys.exit(app.exec())
```

File Analysis: gui.py

- **Purpose:** GUI for real-time QKD data visualization
- **Key Features:**
 - Tabbed interface: Overview, SPD1, SPD2, QBER, Throughput, Visibility, SPD1 Decoy
 - Plots: Histograms (40 bins, 0–4000 ps), line graphs (30s window)
 - Interactive: Tooltips, marquee, mode switching
- **Code Snippet (Plot Update):**

```
1 def update_plots(self):
2     try:
3         for _ in range(50):
4             data = self.data_queue.get_nowait()
5             current_time = time.time() - self.start_time
6             if data['type'] == 'qber':
7                 qber_val = float(data['value'])
8                 self.qber_x_all.append(current_time)
9                 self.qber_y_all.append(qber_val)
10                self.qber_line_all.setData(self.qber_x_all, self.qber_y_all)
```

File Analysis: data_processor.py

- **Purpose:** Acquires and parses QKD data
- **Key Features:**
 - Modes: File (output.txt) or console (UART)
 - Parses session number, timestamps, QBER, keys, etc.
 - Threaded, queue-based data transfer
- **Code Snippet (Parsing):**

```
1 def parse_and_queue(self, line: str):
2     if line.startswith("QBER_VALUE_IS:"):
3         value = float(line.split(':')[1])
4         self.data_queue.put({"type": "qber", "value": value})
5         self.last_session_data["qber"] = value
6         self.session_data_types.add("qber")
```

File Analysis: uart_read.cpp

- **Purpose:** Reads UART data and writes to test.txt
- **Key Features:**
 - Configures serial port (115200 baud, 8-N-1)
 - Reads data into 256-byte buffer
 - Outputs to test.txt and stdout
- **Code Snippet (Serial Read):**

```
1 void readData(HANDLE hport, FILE *fp2) {
2     char buffer[256];
3     DWORD bytesRead;
4     while (1) {
5         if (ReadFile(hport, buffer, sizeof(buffer), &bytesRead, NULL)) {
6             if (bytesRead > 0) {
7                 for (DWORD i = 0; i < bytesRead; i++) {
8                     printf("%c", buffer[i]);
9                     fprintf(fp2, "%c", buffer[i]);
10                    fflush(fp2);
11                }
12            }
13        }
14    }
15 }
```

Data Flow and Interaction

- ① `uart_read.cpp` reads serial data, writes to `test.txt` and also to console simultaneously.
 - ② `data_processor.py` reads `test.txt` (file mode) or UART (console mode)
 - ③ Parsed data queued to `gui.py` via Queue
 - ④ `gui.py` updates plots every 20 ms
 - ⑤ `main.py` coordinates initialization and mode selection
- **Session Management:** Synchronizes on `SESSION_NUMBER`, handles missing data
 - Plots **previous session** data for missing data in **current session**
 - **previous session** data is maintained using list

Visualization Features

- **Tabs:** Overview, SPD1/SPD2 Histograms, QBER, Throughput, Visibility, SPD1 Decoy
- **Plots:**
 - Histograms: 40 bins (0–4000 ps, each bin of size 100)
 - Line Plots: 30s window, dynamic y-axis (based on current window min and max value)
- **Interactivity:** Tooltips, marquee, mode toggle

Screenshots (Overview)



Figure: Overview of entire GUI

Screenshots (Plot_Overview)



Figure: plotting Overview of entire GUI

Screenshots (Individual_Plot_View)



Figure: Individual Plot View

Screenshots (File_Mode_View)



Figure: File Mode View

Screenshots (Console_Mode_View)



Figure: Console Mode view

Screenshots (Full_Key_On_Hover)



Figure: Display of entire key on hover

Challenges and Solutions

- **Challenge:** Real-time data processing
 - **Solution:** Threaded DataProcessor, queue-based transfer
- **Challenge:** Session data consistency
 - **Solution:** Default values for missing data, session tracking
- **Challenge:** GUI responsiveness
 - **Solution:** PyQt6 with 20 ms update interval

Conclusion

- **Summary:** A robust tool for real-time QKD data visualization.
- **Importance:** Enables efficient monitoring and analysis of quantum key distribution data.
- **Next Steps:** Testing with real hardware, deployment, feature expansion

Thank you!

Any questions?