Quantum Key Distribution (QKD) GUI

MANIKANT KUMAR - (PIP-42)
Under the guidance of
XYZ

July 28, 2025

Contents

1 Introduction

The Quantum Key Distribution (QKD) Analyzer is a Python-based desktop application designed to provide an intuitive graphical user interface (GUI) for visualizing and analyzing QKD data. The GUI, built with `PyQt6`, serves as a user-friendly dashboard, presenting real-time graphs of key metrics like **QBER**, **Throughput**, and **Timestamps**. Using `pyqtgraph` for dynamic plotting, it offers interactive tabs, control buttons, and a status bar, making it accessible to users of all skill levels.

The GUI features multiple tabs to display metrics such as **QBER** (error rates), **Throughput** (key generation speed), and **Timestamps** (photon detection times at `SPD1` and `SPD2`). Users can interact with buttons like *Start*, *Stop*, *Resume*, and *Toggle Mode* to manage data processing in *File Mode* (reading `build/output.txt`) or *Console Mode* (live program output). A status bar displays information like `Current Session: 10`, ensuring users can monitor the application's state effortlessly.

1.1 Purpose

The QKD Analyzer GUI aims to:

- **Visualize Data**: Present metrics like **QBER** and **Throughput** in interactive graphs across multiple tabs.
- **Support Multiple Sources**: Process data from `build/output.txt` (*File Mode*) or live program output (*Console Mode*).
- **Handle Missing Data**: Use previous session values to maintain continuous graphs.
- **Provide Intuitive Controls**: Offer buttons like *Start* and *Stop* for easy operation.
- **Ensure Accessibility**: Create a user-friendly interface with clear tabs and status updates for all users.

1.2 System Requirements

To run the QKD Analyzer GUI:

- **Operating System**: Windows, Linux, or macOS.
- **Python**: Version 3.6+, available at `https://www.python.org`.
- **Python Libraries**:
  - `PyQt6`: For the graphical interface and controls.
  - `pyqtgraph`: For real-time, interactive plotting.
  - `numpy`: For numerical calculations.
- **Input Data**:
  - *File Mode*: A `build/output.txt` file with QKD data.

    – *Console Mode*: A program outputting QKD data.

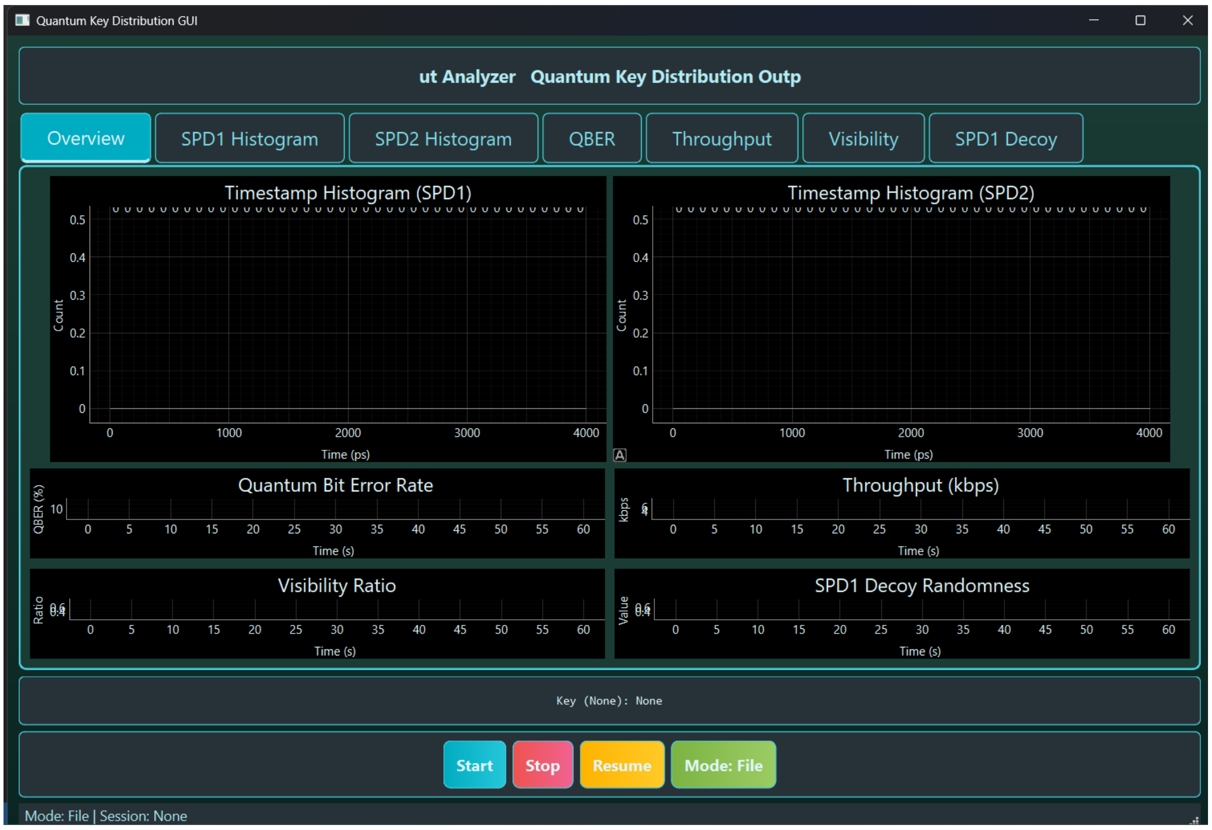
- **Storage**: A few megabytes for files.
- **Memory**: At least 512 MB RAM.

2 Understanding QKD Data Metrics

The QKD Analyzer GUI visualizes key metrics essential for analyzing QKD systems. These metrics, displayed in interactive graphs, include:

- **Timestamps**: Photon detection times at `SPD1` and `SPD2` (0 to 4000 picoseconds).
- **QBER (Quantum Bit Error Rate)**: Error percentage in the key (e.g., 2% is good).
- **Throughput**: Key generation speed (kbps).
- **Visibility**: Signal quality (0 to 1).
- **Decoy Randomness**: Random values (0 to 1) for security.
- **Key Bits**: The variable size key(128/256/512...bits) (e.g., `0101...`).

For example, a **QBER** above 5% may indicate issues, while **Visibility** of 0.95 suggests good signal quality. The GUI's tabs make these metrics easy to monitor.



Key (None): None

Mode: File | Session: None

Figure 1: QKD Analyzer GUI Overview

3 How the Application is Built

- Overview: The QKD Analyzer uses three Python files (`main.py`, `gui.py`, `data_processor.py`) to read, process, and display QKD data via a GUI.
- Purpose: Enable real-time visualization of metrics like **QBER** and **Timestamps** using `PyQt6` and `pyqtgraph`.
- Design: Modules communicate via a `Queue`, like a conveyor belt, moving data from processing to display.

3.1 Component Overview

- **`main.py`**:
  - Entry point for the application.
  - Parses command-line arguments (e.g., `python main.py file` for *File Mode*, `python main.py console` for *Console Mode*).
  - Creates a `Queue` to hold data packets.
  - Initializes `DataProcessor` to read and process data.
  - Launches `MainWindow` for the GUI using `PyQt6`.
- **`gui.py`**:
  - Defines `MainWindow` for the GUI interface.
  - Builds tabs (e.g., `Overview`, `QBER`) with `PyQt6` widgets.
  - Displays `pyqtgraph` plots for metrics like **QBER** and **Timestamps**.
  - Includes `Control Buttons` (*Start*, *Stop*, *Resume*, *Toggle Mode*).
  - Shows `Status Bar` with session info (e.g., `Current Session: 10`).
  - Retrieves data from `Queue` to update plots.
- **`data_processor.py`**:
  - Handles data reading from `build/output.txt` (*File Mode*) or program output (*Console Mode*).
  - Parses metrics (e.g., **QBER**, **Timestamps**..etc) from lines like `SPD1_QBER_VALUE_IS:2.34`.
  - Creates `dict` packets (e.g., `{"type": "qber", "value": 2.34}`) for the `Queue`.
  - Runs in a separate thread to avoid slowing the GUI.

3.2 Data Flow

- Steps:
  - `main.py` starts, creating `Queue`, `DataProcessor`, and `MainWindow`.
  - `DataProcessor` reads data from `build/output.txt` (*File Mode*) or program output (*Console Mode*).
  - `DataProcessor` parses lines (e.g., `SPD1_QBER_VALUE_IS:2.34` to **QBER** = 2.34).
  - Parsed data is organized by `SESSION_NUMBER` (e.g., `SESSION_NUMBER:5`).
  - Data is sent to `Queue` as `dict` packets (e.g., `{"type": "qber", "value": 2.34}`)
  - `MainWindow` retrieves packets from `Queue` and updates `pyqtgraph` plots (e.g., **QBER** plot).
- Example: `SPD1_VALUES:1234` becomes `{"type": "timestamp_spd1", "value": 1234}`, updating the `SPD1` histogram.

```python
# In data_processor.py
from queue import Queue
queue = Queue()
queue.put({"type": "qber", "value": 2.34})

# In gui.py
while not queue.empty():
    data = queue.get()
    if data["type"] == "qber":
        update_qber_plot(data["value"])
```

3.3 Threading for Performance

- Purpose: Keep GUI responsive during data processing.
- Threads:
  - **Data Reading Thread**: `data_processor.py` runs separately to read and parse data (e.g., `build/output.txt` or console output).
  - **GUI Thread**: `gui.py` handles user interactions (e.g., *Start* button) and plot updates in `MainWindow`.
- Synchronization:
  - `Queue` connects threads, allowing `data_processor.py` to send data and `gui.py` to receive it.
  - Thread-safe design prevents data conflicts.
- Benefit: Prevents GUI freezing when processing large files (e.g., 1 MB `output.txt`).

Placeholder: Insert a diagram showing data from source to `Queue` to `MainWindow`.

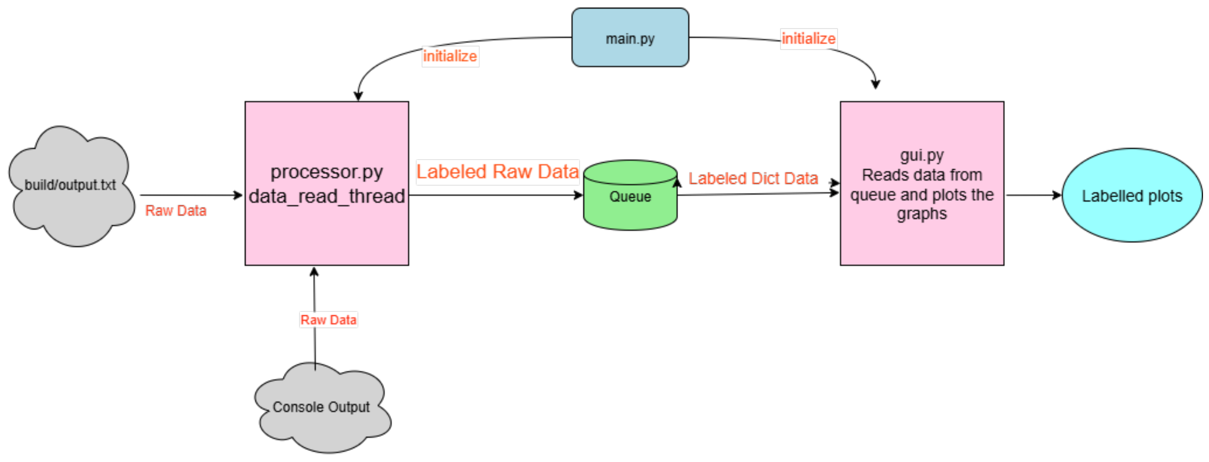


Figure 2: Data Flow in the QKD Analyzer

4 Exploring the User Interface

The QKD Analyzer's GUI enables easy monitoring of QKD data via tabs, buttons, and a status bar, suitable for all users.

4.1 Main Window Layout

The `MainWindow` includes:

- **Tabs**: Six tabs for metrics:
  - *Overview*: Shows `SPD1/SPD2` **Timestamps**, **QBER**, **Throughput**, **Visibility**, **Decoy Randomness**, and key display.
  - *SPD1 Timestamps*: Histogram of `SPD1` times.
  - *SPD2 Timestamps*: Histogram for `SPD2`.
  - *QBER*: Line graph for error rates.
  - *Throughput*: Line graph for key speed.
  - *Visibility*: Line graph for signal quality.
  - *SPD1 Decoy Randomness*: Line graph for randomness.
- **Key Display**: Shows bits of variable size key (e.g.128/256/512...., `010101...`); hover for full key.
- **Control Buttons**:
  - *Toggle Mode*: Switches between *File Mode* (`output.txt`) and *Console Mode*.
  - *Start*: Begins processing.
  - *Stop*: Pauses, saves file position (*File Mode*).
  - *Resume*: Continues (*File Mode*).
- **Status Bar**: Shows `Current Session:x` and `Mode:console/file`.

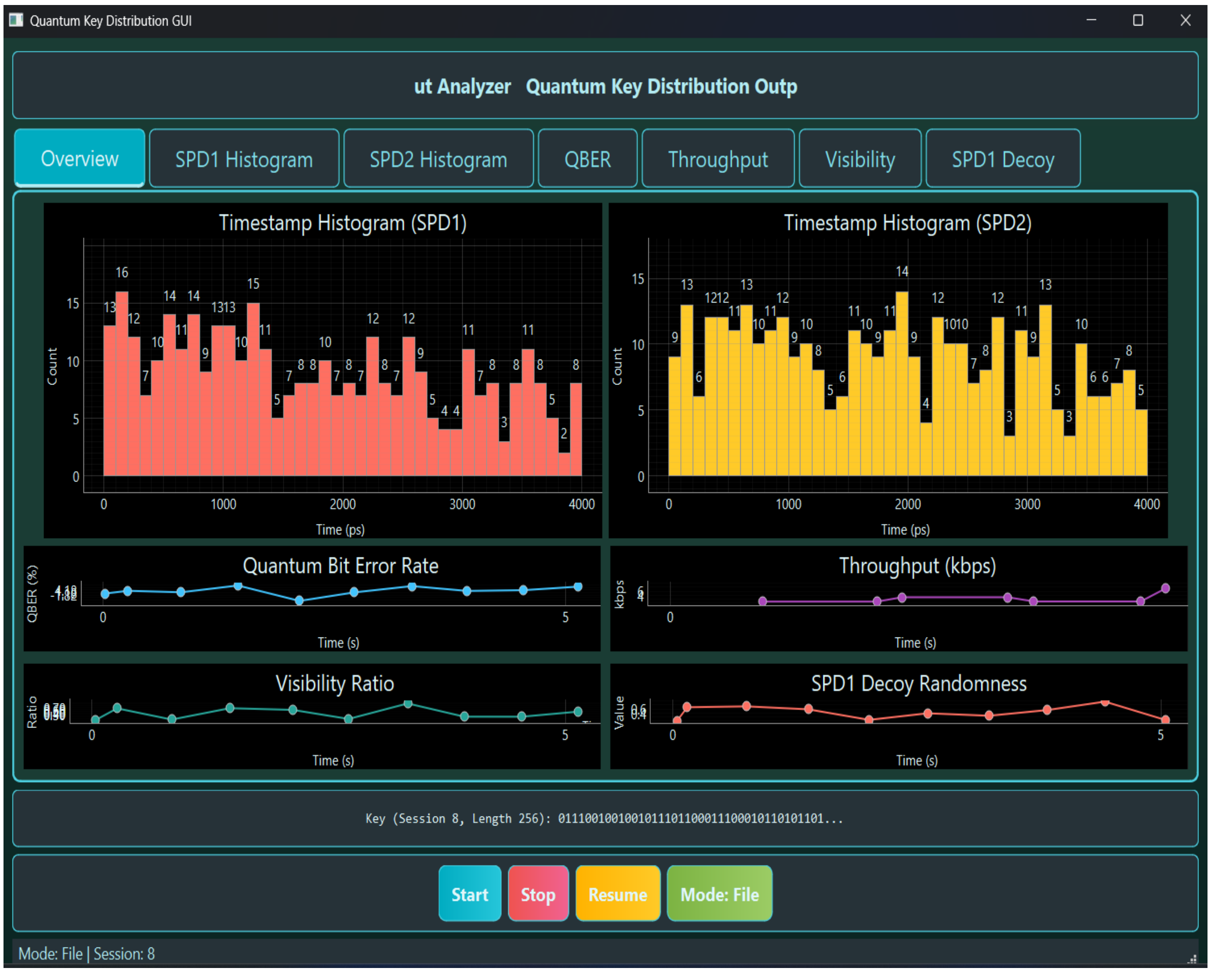
For example: a user clicks *Start* in *File Mode* and sees graphs in *Overview*.

4.2 Understanding the Graphs

Each graph provides insight:

1. **SPD1 Timestamps Histogram**:
   - *What it shows*: `SPD1` times (0 to 4000 ps) in 40 bars.
   - *Y-Axis*: Photon count, scales to 20% above tallest bar or 10.
   - *X-Axis*: 0 to 4000 ps.
   - *Example*: 50 photons at 1000 ps create a bar at 50.
   - *Why it matters*: Consistent **Timestamps** ensure key reliability.
2. **SPD2 Timestamps Histogram**:
   - *What it shows*: Same for `SPD2`.
   - *Example*: Differing histograms suggest misalignment.
3. **QBER Plot**:
   - *What it shows*: **QBER** (%) over time.
   - *Y-Axis*: Fits data with 10% margin or 0 to 20%.
   - *X-Axis*: Last 60 seconds.
   - *Example*: **QBER** of 3% is stable; 10% suggests issues.
4. **Throughput Plot**:
   - *What it shows*: Key speed (kbps).
   - *Y-Axis*: 0 to 10 kbps.
   - *Example*: 5 kbps is moderate; 1 kbps indicates slowdowns.
5. **Visibility Plot**:
   - *What it shows*: **Visibility** (0 to 1).
   - *Y-Axis*: Fits data with 10% margin or 0 to 1.
   - *Example*: 0.95 is excellent; 0.6 indicates interference.
6. **SPD1 Decoy Randomness Plot**:
   - *What it shows*: **Decoy Randomness** (0 to 1).
   - *Y-Axis*: 0 to 1.
   - *Example*: Fluctuations around 0.5 show good randomness.

For example, a user sees a **QBER** spike in *Overview*, switches to *QBER*, and checks `Current Session`. **Placeholder**: Insert a screenshot of the *Overview* tab.



```
Key (Session 8, Length 256): 0111001001001011101100011100010110101101...
```

```
Mode: File | Session: 8
```

Figure 3: GUI Overview Tab with Graphs

4.3 User Interaction

The GUI is intuitive:

- **Navigating Tabs**: Click tabs (e.g., *QBER*).
- **Using Buttons**: Click *Start*, *Stop*, *Resume* (*File Mode*), or *Toggle Mode*.
- **Status Bar**: Shows `Current Session:x` **and** `Mode:console/file`.
- **Key Display**: Shows bits of variable size key (e.g.128/256/512...., `010101...`); hover for full key.

For example, a user pauses after high **QBER**, checks the key, and resumes.

5 How the Application Works

The QKD Analyzer reads, processes, and displays QKD data in real-time via the GUI.

5.1 Getting Data

Two modes:

- ***File Mode***: Reads `build/output.txt`, ideal for testing.
- ***Console Mode***: Reads live program output, for experiments.

Data format:

- `INPUT_STRING:test`: Optional (*Console Mode*).
- `SESSION_NUMBER:5`
- `SPD1_VALUES:`: Timestamps (e.g., `1234`).
- `DECOY_STATE_RANDOMNESS_AT_SPD1:0.75`
- `SPD2_VALUES:`: Timestamps.
- `VISIBILITY_RATIO_IS:0.92`
- `SPD1_QBER_VALUE_IS:2.34`
- `KEY_BITS:0101...`
- `KEY_RATE_PER_SECOND_IS:5.67`

Example `output.txt`:

```
SESSION_NUMBER:1
INPUT_STRING:test
SPD1_VALUES:
1234
5678
DECOY_STATE_RANDOMNESS_AT_SPD1:0.65
SPD2_VALUES:
2345
6789
VISIBILITY_RATIO_IS:0.88
SPD1_QBER_VALUE_IS:3.21
KEY_RATE_PER_SECOND_IS:4.50
```

5.2 Processing Data

The `data_processor.py` file:

- **Reading**: Opens `build/output.txt` or captures output.
- **Parsing**: Extracts values (e.g., `SPD1_QBER_VALUE_IS:2.34` yields **QBER** = 2.34).
- **Organizing**: Groups by `SESSION`$_{N}UMBER.$**`Handling Missing Data`** :
- - First session: Defaults (**QBER** = 0, key = `0*128`).

- Later sessions: Reuses previous values.

Sending: Queues packets like `{"type": "qber", "value": 2.34}`.
For example, `SPD1_VALUES:1234` becomes `{"type": "timestamp_spd1", "value": 1234}`.

5.3 Updating Graphs

The `gui.py` file:

- **Queue Checking**: Checks `Queue` every 0.1 seconds via `PyQt6`.
- **Histogram Updates**: Assigns timestamps to 40 bins (0–4000 ps).
- **Line Graph Updates**: **QBER**, **Visibility**, **Throughput**, and **Decoy Randomness** use dynamic y-axes, adjusting automatically to fit data values with a 10% margin above and below the minimum and maximum. In `gui.py`, the `MainWindow` dequeues `dict` packets from the `Queue` (e.g., `{"type": "qber", "value": 2.34}`), appends values to a time-series array, updates the `pyqtgraph.PlotWidget` with new data, and sets the y-axis range dynamically based on the data's min/max values.
- **Key Display**: Shows variable size key bits.

For example, `{"type": "qber", "value": 3.21}` adds 3.21% to **QBER**.

5.4 Real-Time Updates

Real-time updates:

- **Timer**: `PyQt6` checks `Queue` every 0.1 seconds.
- **Large Datasets**: Processes chunks to manage memory.

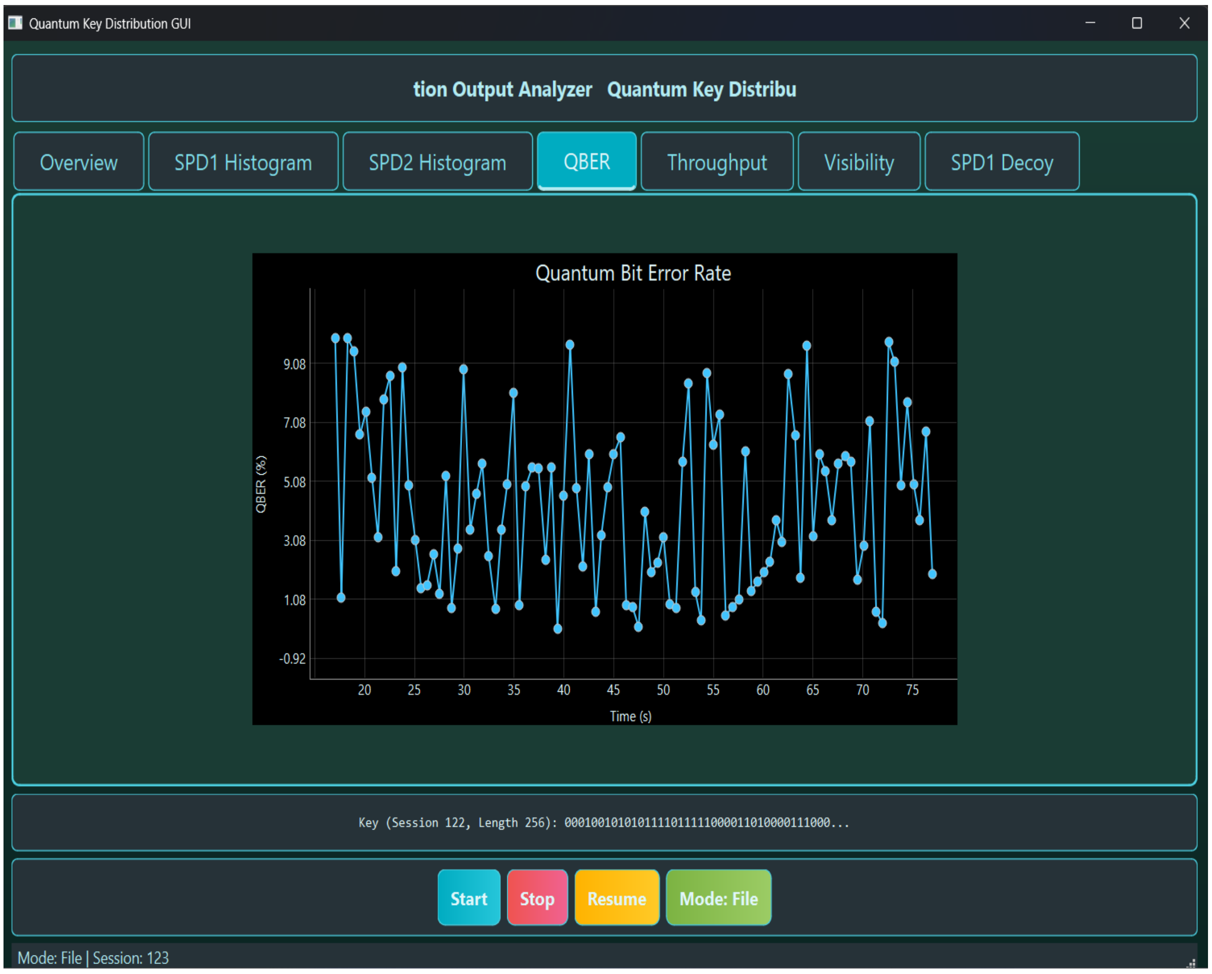
- **Error Handling**: Skips malformed lines (e.g., `SPD1`$_{Q}BER_{V}ALUE_{I}S$: abc).**`Example`** $: 100sessionsupdategraphs600timesperminute$. Troubleshooting:

- No updates: Check `Queue` via `logging.DEBUG` in `data_processor.py`.
- Slow updates: Reduce console output or file size.

5.5 Controlling the Application

Buttons:

- *Toggle Mode*: Switches modes, clears graphs.
- *Start*: Begins processing.
- *Stop*: Pauses, saves position (*File Mode*).
- *Resume*: Continues (*File Mode*).



```
Key (Session 122, Length 256): 0001001010101111011111000011010000111000...
```

Mode: File | Session: 123

Figure 4: GUI QBER Plot

6 Installation and Setup

Steps:

6.1 Installing Python and Libraries

1. **Install Python**:
   - Download Python 3.6+ from `https://www.python.org`.
   - Run `python -version`.
2. **Install Libraries**:
   - Run:

```
pip install PyQt6 pyqtgraph numpy
```

3. **Prepare Input Data**:
   - *File Mode*: Create `build/output.txt`.
   - *Console Mode*: Ensure program outputs QKD data.

4. **Save Python Files**:
   - Save `main.py`, `gui.py`, `data_processor.py` in src folder.
5. **Run**:
   - By using command **Python main.py** and then in GUI choose mode and also provide input for console mode.

6.2 Creating a Standalone Executable

Use `PyInstaller`:

1. **Install PyInstaller**:
   - Run:

     ```
     pip install pyinstaller
     ```

2. **Create Executable**:
   - Navigate to folder with `main.py`, `gui.py`, `data_processor.py`.
   - Run:
     - Single file:

       ```
       pyinstaller --onefile --name QKDAnalyzer main.py
       ```

     - Folder:

       ```
       pyinstaller --name QKDAnalyzer main.py
       ```

   - Outputs: `dist/QKDAnalyzer.exe` (Windows) or `QKDAnalyzer` (Linux/macOS).
3. **Customize**:
   - Icon (Windows):

     ```
     pyinstaller --onefile --name QKDAnalyzer --icon=app.ico main
        .py
     ```

   - Hide console:

     ```
     pyinstaller --onefile --name QKDAnalyzer --windowed main.py
     ```

4. **Run Executable**:
   - *File Mode*: Copy executable in build where the output.txt is present and run by simple double clicking on executable.
   - *Console Mode*: Ensure program is accessible.
   - Run `QKDAnalyzer.exe` or `./dist/QKDAnalyzer/QKDAnalyzer`.

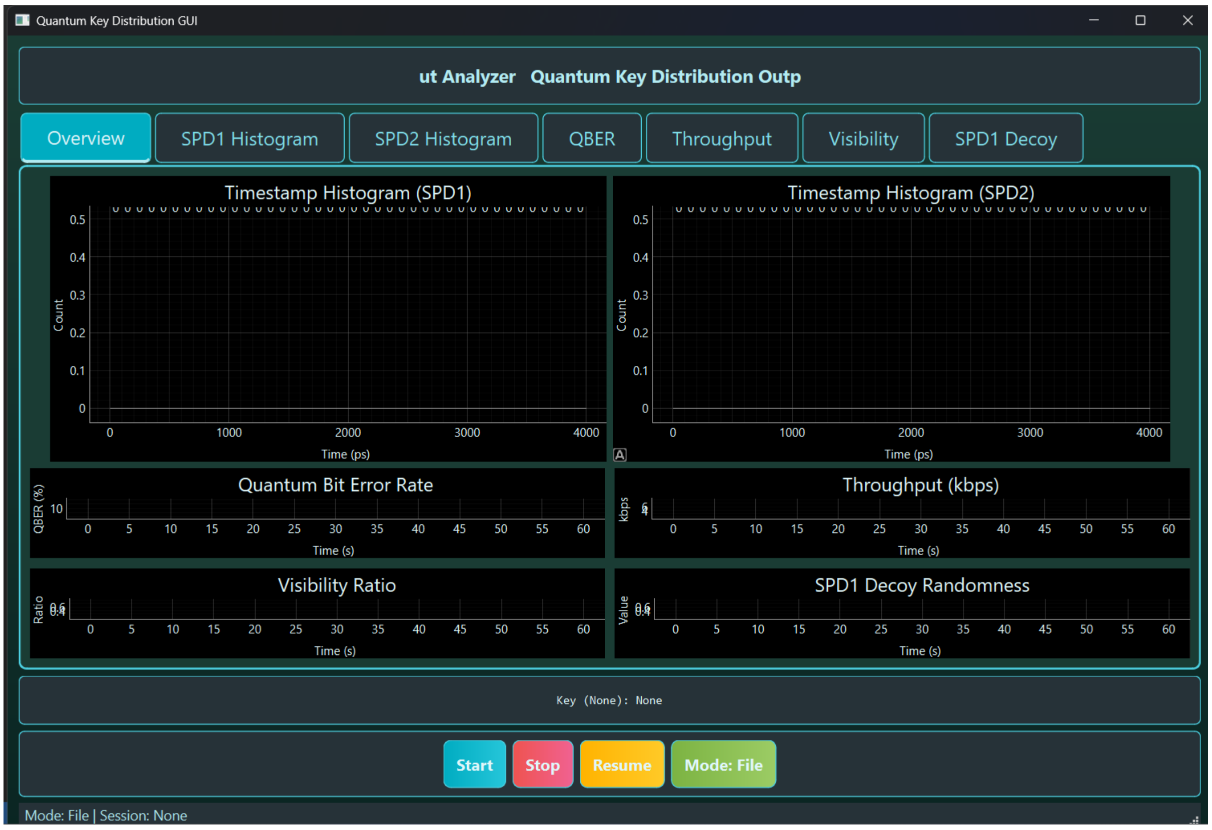
5. **Troubleshooting**:
    - Missing dependencies: Run `pyinstaller -clean`.
    - *File Mode* errors: Verify `build/output.txt`.
    - Debug: Add `-log-level DEBUG`.

7 Using the QKD Analyzer

Steps:

1. **Open**: Run `python main.py file` **or** `python main.py console`, **or** `QKDAnalyzer.e`
2. **Choose Mode**: Click *Toggle Mode*.
3. **Start**: Click to start.
4. **Stop**: Click to stop(saves current file position in file mode).
5. **Resume**: Resumes from current file poition (file mode only).
6. **View**: Use *Overview* or tabs; hover over key to see full key.
7. **Pause/Resume**: Click *Stop*, *Resume* (*File Mode*).

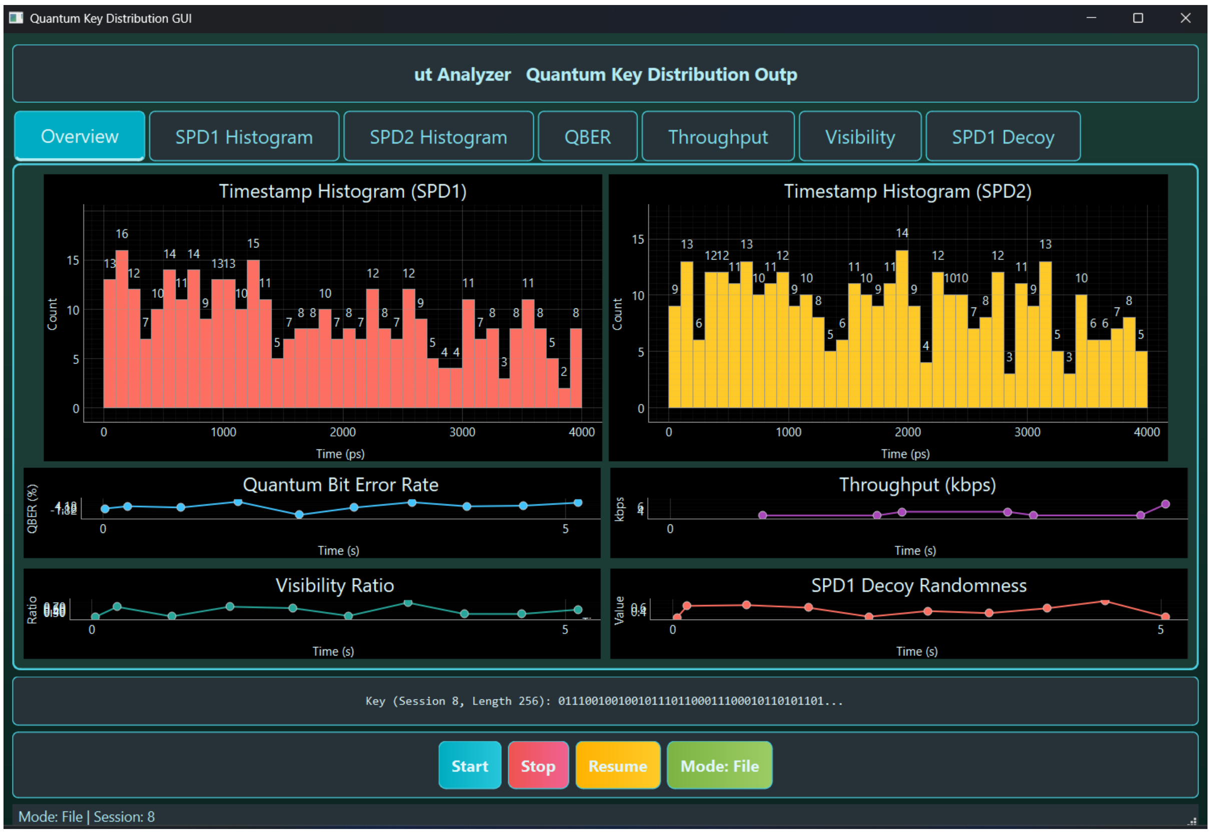
8 SCREENSHOTS OF VARIOUS COMPONENTS



```
Key (None): None
```

```
Mode: File | Session: None
```

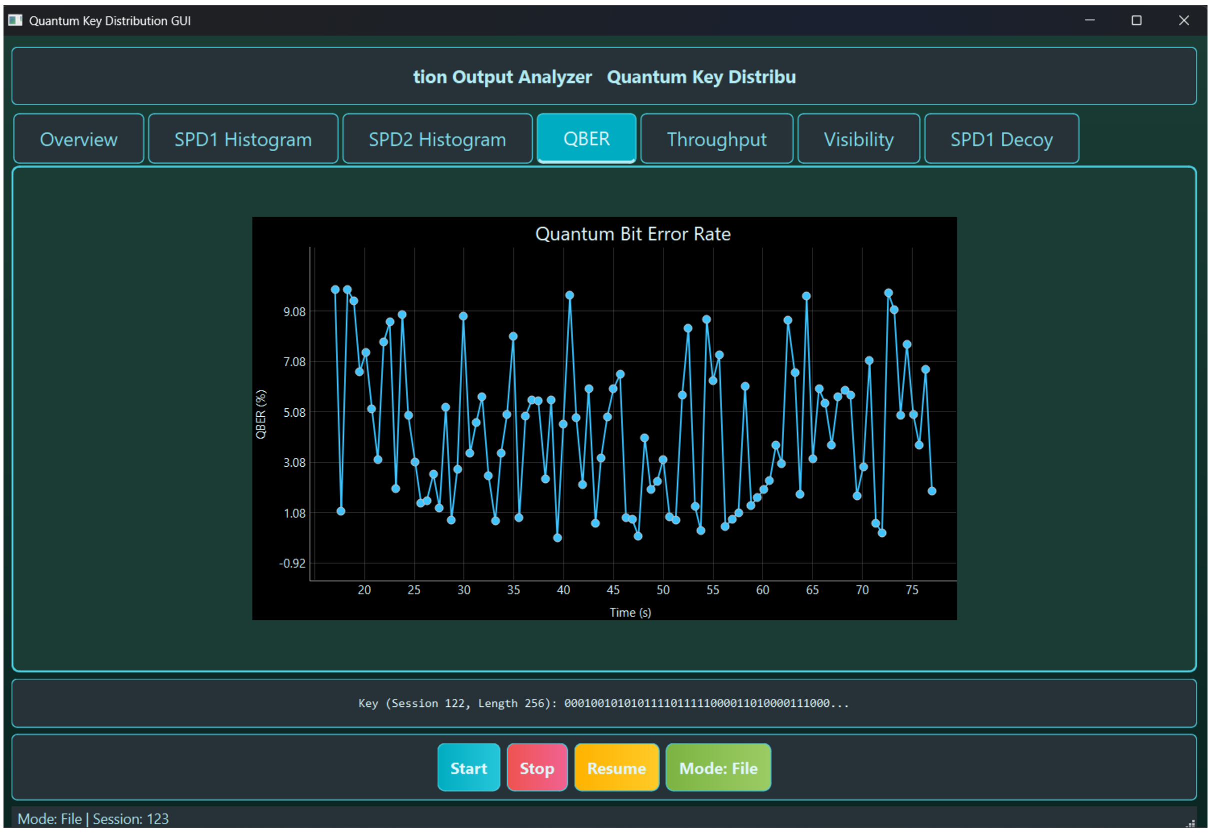
Figure 5: QKD Analyzer GUI Overview



```
Key (Session 8, Length 256): 0111001001001011101100011100010110101101...
```

```
Mode: File | Session: 8
```

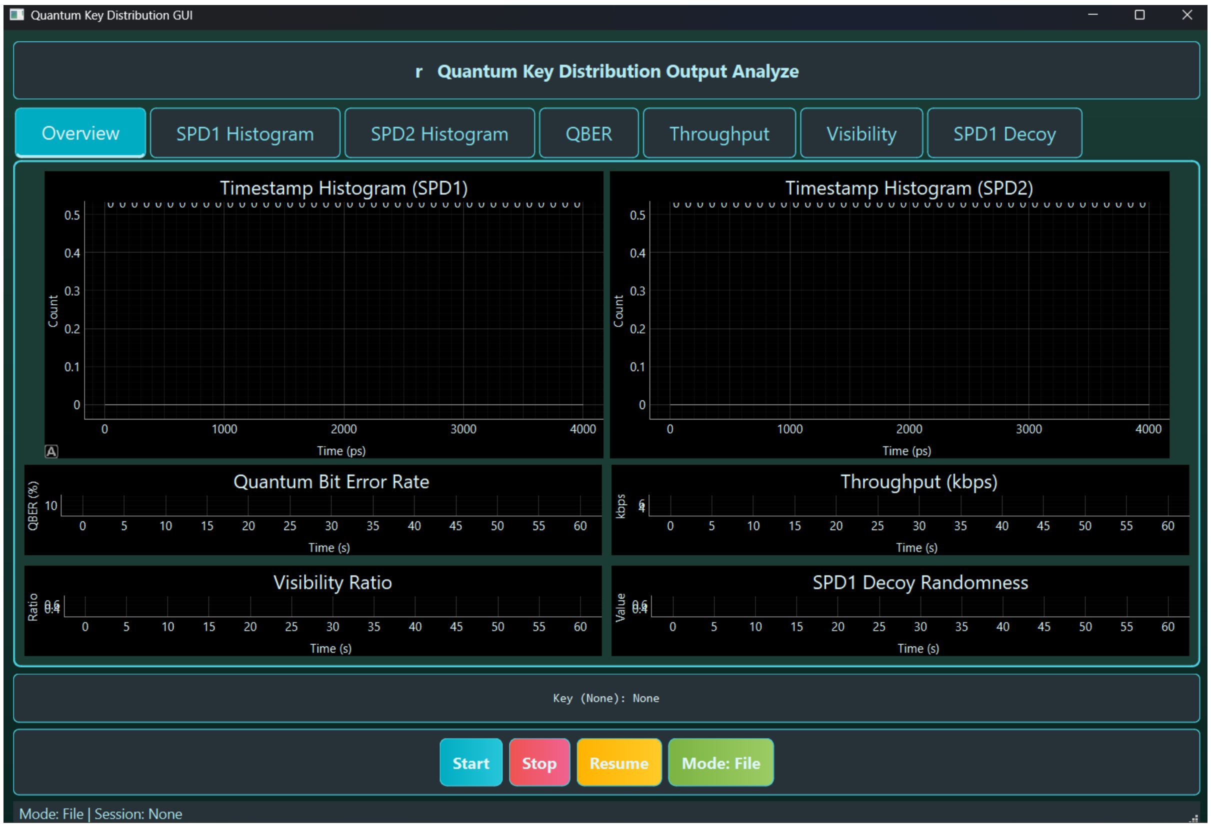
Figure 6: QKD Analyzer GUI Overview with plots



Key (Session 122, Length 256): 0001001010101111011111000011010000111000...

Mode: File | Session: 123

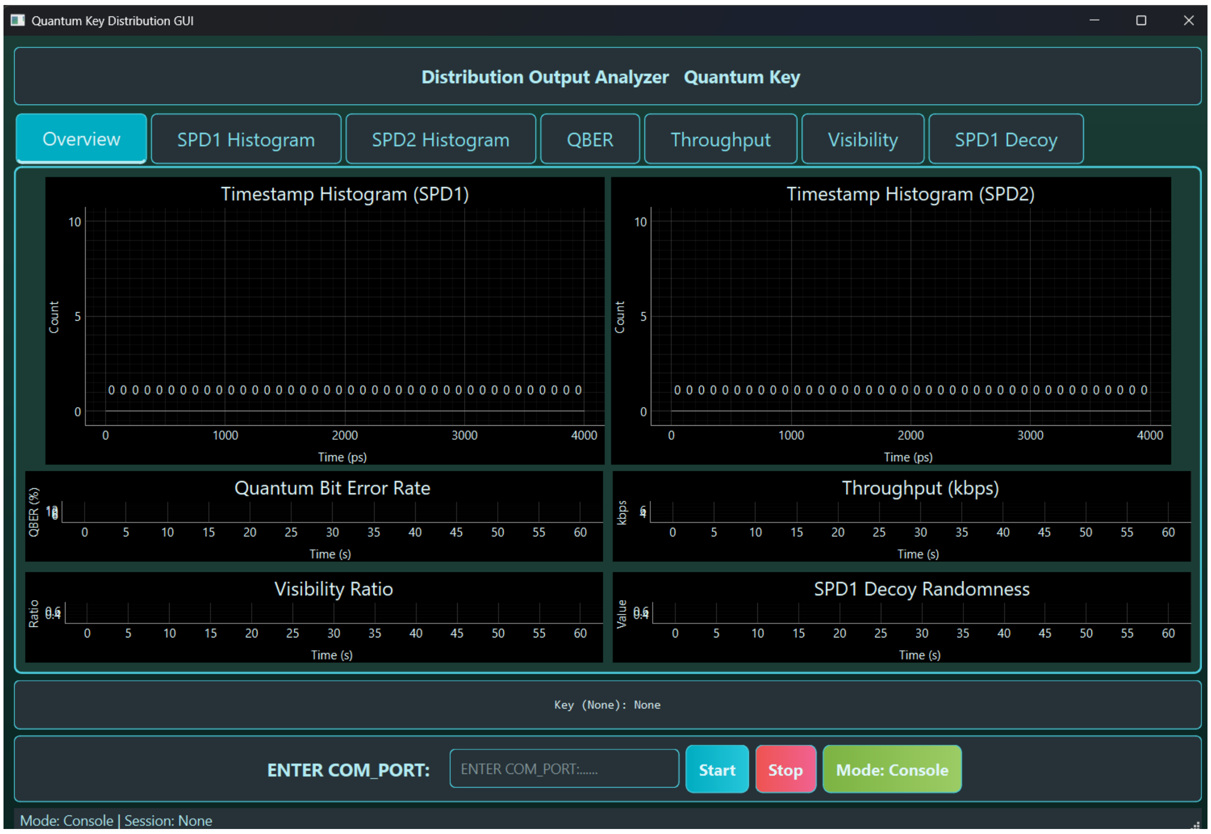
Figure 7: QBER plot for individual tab view



Key (None): None

Mode: File | Session: None

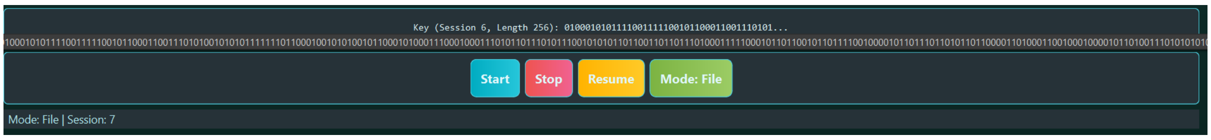
Figure 8: Vies in file mode



```
Key (None): None
```

Mode: Console | Session: None

Figure 9: Vies in console mode



```
Key (Session 6, Length 256): 0100010101111001111100101100011001110101...
1000101011110011111001011000110011101010010101011111110110001001010100101100010100011100010001110101101110101110010101011011001101101110100011111000101101100101101111001000010110111011010110110000110100011001000100001011010011101010101
```

Mode: File | Session: 7

Figure 10: Full key bits on hover