A Modular Simulator for Quantum Key Distribution Protocols: BB84, DPS, and COW

Anurag Chaudhary, Jayesh Patil, Aditi Das
Supervisor Dr. Ankur Raina
IISER Bhopal
anuragchaudharym87@gmail.com, jayesh20157@gmail.com,
aditi.das1601@gmail.com

Abstract

We present a modular and extensible simulation platform for Quantum Key Distribution (QKD) protocols, supporting BB84, Differential Phase Shift (DPS), and Coherent One-Way (COW) schemes. The simulator integrates a Python FastAPI backend and a React-based frontend to provide interactive, multi-node, and multi-channel simulations. It allows protocol-specific configuration and visualizes quantum parameters including QBER and secure key rate. This work aims to bridge theoretical QKD with practical architectural insights.

I. Introduction

Quantum Key Distribution (QKD) is a revolutionary cryptographic technique leveraging quantum mechanical principles to enable unconditional security in communication. Unlike classical key exchange systems, the security of QKD stems from the properties of quantum systems — notably, the no-cloning theorem and measurement-induced disturbance. Various QKD protocols exist, such as BB84, DPS, and COW, each offering unique trade-offs in terms of implementation complexity, security model, and key rate efficiency. Our simulation platform aims to make these protocols accessible for both researchers and students by modeling their physics and digital logic.

II. QKD Protocols: Theory and Implementation

A. BB84 Protocol

The BB84 protocol, proposed by Bennett and Brassard in 1984, is the first and most widely studied QKD protocol. The steps are as follows:

- **Encoding:** Alice uses a quantum random number generator (QRNG) to select a random bit (0 or 1) and a basis: rectilinear ($\{|0\rangle, |1\rangle\}$) or diagonal ($\{|+\rangle, |-\rangle\}$).
- **Transmission:** She sends the qubit to Bob over a quantum channel.
- **Measurement:** Bob randomly chooses a basis for measurement.
- **Sifting:** Over a classical authenticated channel, they disclose basis choices and keep bits where the bases matched.

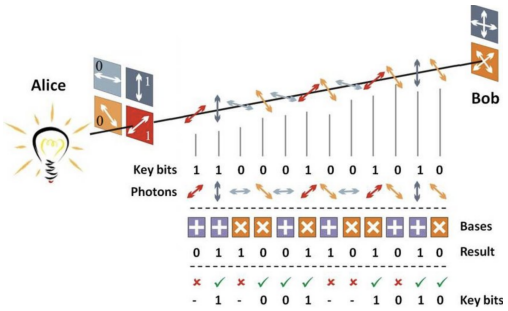

Fig. 1. BB84 basic scheme

- **Post-Processing:** Includes error correction and privacy amplification to create a shared secret key.
- **Implementation:** The BB84 protocol is implemented in the generate_and_share_key_bb84 method of the Node class (see simulation/Network.py). Bit and basis choices are generated using Python's random module. Sifting, error calculation, and post-processing are handled by the calculate_qber and postprocessing functions, which are imported and called in both api.py (for the web API) and main.py.
- **User-configurable parameters:** such as the number of pulses (num_pulses), pulse repetition rate (pulse_repetition_rate), and detector efficiency are passed from the frontend (via the API endpoint in api.py) or from command-line arguments in main.py, and are then forwarded to the simulation functions and classes.

Security: Any attempt by an eavesdropper (Eve) introduces detectable errors due to the no-cloning theorem. Eve cannot copy or measure the qubits without disturbing them.

B. Differential Phase Shift (DPS) Protocol

DPS-QKD simplifies the setup by removing basis choice. Its operation involves:

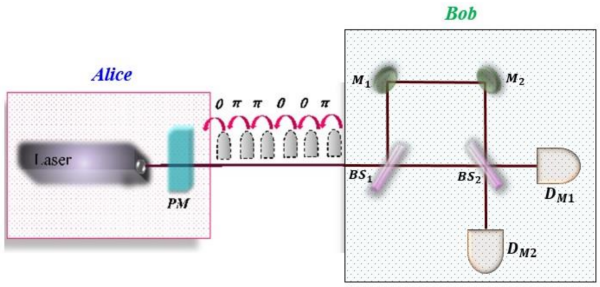

Fig. 2. (Color Online) Illustration of DPS protocol. PM: phase modulator, BS1, BS2 are beamsplitters and M1, M2 are mirrors.

- **Encoding:** Alice generates a series of weak coherent pulses with equal intensity. A phase modulator imparts a phase shift of 0 or π to each pulse.
- **Transmission:** Pulses are sent consecutively through an optical fiber.

- **Interference:** Bob uses a delay-line Mach-Zehnder interferometer that interferes pulse n with pulse $n-1$.
- **Detection:** Depending on the phase difference, either detector A (0 phase difference) or detector B (π phase) clicks.
- **Sifting:** Bit value is determined by which detector clicked. Bob sends timestamps of detections to Alice.
- **Implementation:** The DPS protocol logic is encapsulated in the generate_and_share_key_dps method of the Node class. Phase encoding and detection are simulated using arrays to represent pulse sequences and their phase shifts. The Mach-Zehnder interferometer and detection events are modeled algorithmically within this method.
- **User-configurable parameters:** such as phase flip probability (phase_flip_prob), number of pulses (num_pulses), and pulse repetition rate are provided by the user through the frontend form handled in frontend/src/components/QKDForm.js and sent to the backend via the /simulate endpoint in api.py),arguments in main.py. These parameters are then passed to the relevant simulation functions.

Security: Resistant to photon-number-splitting (PNS) attacks. The detection event is distributed over multiple pulses, making it harder for Eve to extract useful information.

C. Coherent One-Way (COW) Protocol

COW-QKD uses temporal encoding with a simple setup. The main elements are:

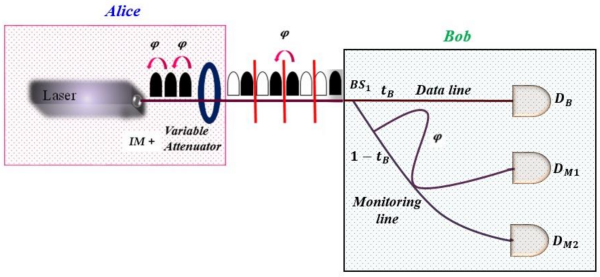

Fig. 3. (Color Online) Illustration of COW protocol. IM: intensity modulator, BS1: beamsplitter

- **Encoding:** Alice sends sequences of two time bins. A logical '0' is encoded as pulse–vacuum (μ–0), and a logical '1' as vacuum–pulse (0–μ). Decoy sequences (μ–μ) are also inserted for monitoring.
- **Detection:** Bob uses a time-of-arrival detector to determine bit values. He also has an interferometer to monitor coherence.
- **Monitoring:** Decoy sequences are used to test coherence. Any eavesdropping attempt disturbs the interference visibility.
- **Implementation:** The COW protocol logic is encapsulated in the generate_and_share_key_cow method of the Node class. The method simulates the creation of data and decoy pulse sequences, time-of-arrival detection for key extraction, and coherence monitoring using an

interferometric measurement. Arrays are used to represent the pulse train and detection events.

- **User-configurable parameters:** such as monitor pulse ratio (cow_monitor_pulse_ratio), extinction ratio (cow_extinction_ratio_db), and detection threshold are set by the user in the frontend (via the QKD form)and are passed to the backend through the API (api.py) or directly to the simulation functions in main.py. These parameters are then used to control the simulation logic and output.

Security: The COW protocol relies on the coherence between consecutive pulses. If Eve attempts to intercept and resend the pulses, she disturbs this coherence. Bob monitors visibility using an interferometric detector, and any drop in visibility signals the presence of an eavesdropper.

III. Hardware Components

Our simulator models the key components involved in realistic QKD systems. Each component emulates essential physical behavior and imperfections to help learners understand their practical roles.

- **Weak Coherent Source (WCS):**
  In most QKD implementations, especially BB84, DPS, and COW, true single-photon sources are replaced with weak coherent sources (WCS). These are laser diodes heavily attenuated such that the average number of photons per pulse, denoted by μ, is significantly less than 1. The emission of photons follows a Poisson distribution:

$$P(n) = \frac{\mu^n}{n!} e^{-\mu}$$

  where n is the number of photons per pulse. Typically, μ ranges between 0.1 to 0.5. This means most pulses are either vacuum or contain only one photon, approximating single-photon behavior while remaining cost-effective and easy to control.
- **Modulators:**
  - **Phase Modulator (PM):**
    Used mainly in the DPS protocol, the phase modulator alters the optical phase of the passing pulse by either 0 or π radians. It utilizes electro-optic crystals (e.g., $LiNbO_3$) whose refractive index changes with applied voltage. This modifies the path length and hence the phase of the transmitted pulse. The QRNG (Quantum Random Number Generator) controls this phase modulation but here we are using Random module of python.
  - **Intensity Modulator (IM):**
    In COW-QKD, the intensity modulator toggles between transmitting a weak coherent pulse (logical pulse) and no pulse (vacuum). This is typically done using a Mach-Zehnder modulator driven by a high-speed signal that controls constructive or destructive interference at the output. IMs are also used to generate decoy states in advanced BB84 variants.

- **Quantum Channel:**
  The quantum channel typically consists of optical fiber which introduces several realistic impairments:
  - **Attenuation:** Light is gradually absorbed or scattered over distance. Standard telecom fiber has attenuation of 0.2 dB/km at 1550 nm.
  - **Dispersion:** Pulses spread temporally due to different frequencies traveling at different speeds.
  - **Phase Noise and Bit Flips:** Modeled by introducing random phase errors or bit-flip probabilities in the simulator.

  The channel also simulates fiber lengths, connector losses, and environmental effects.
- **Interferometers:**
  Interference-based detection is central to both DPS and COW protocols. Our simulator models the physics of:
  - **Mach-Zehnder Interferometer (MZI):** Used at the receiver to interfere consecutive pulses. One arm includes a delay equal to the pulse interval, and constructive or destructive interference reveals the phase difference (DPS) or coherence (COW decoy monitoring).
- **Single-Photon Detectors (SPDs):**
  The final step in any QKD protocol involves detecting the quantum states. The simulator implements SPDs with the following realistic characteristics:
  - **Efficiency (η):** Probability that an incoming photon triggers a detection event. Typical values range from 10–70%.
  - **Dark Counts:** Random clicks due to thermal noise or stray photons, not from actual signals.
  - **Dead Time:** After detecting a photon, the detector needs a short recovery period (tens to hundreds of ns) where it cannot detect again.

  Depending on the protocol, SPDs are used with or without time-resolved detection or basis switching.

All components in our simulator are parameterized. Users can set experimental conditions such as wavelength, mean photon number μ, detector efficiency, dark count probability, and fiber length. This provides flexibility to simulate both ideal and noisy environments, and to study how each hardware imperfection affects the Quantum Bit Error Rate (QBER) and final key rate.

IV. System Architecture

The simulator is structured as follows:

A. Backend (FastAPI + Python)

- Modular simulation logic for all QKD protocols

```
├── api.py               # FastAPI entry point (backend API)
├── main.py              # Backend entry script and simulation runner
├── simulation/          # Core simulation logic (Python package)
│   ├── __init__.py
│   ├── Network.py       # Network and node management
│   ├── Hardware.py      # All hardware components (light source, modulators, channel, etc.)
│   ├── Sender.py        # Sender logic for all QKD protocols
│   ├── Receiver.py      # Receiver logic for all QKD protocols
│   └── ...              # Other simulation files
├── frontend/            # React frontend
│   ├── src/
│   │   ├── components/  # Main React components
│   │   │   ├── QKDForm.js
│   │   │   ├── QKDNetwork.js
│   │   │   └── Results.js
│   │   └── ...          # Other React files
│   └── ...              # Frontend config and public assets
├── requirements.txt     # Python dependencies
└── README.md            # Project documentation
```

Fig. 4. QKD Simulator Directory Structure

- Sender and Receiver modules for protocol execution
- Hardware and channel modeling
- QBER and key rate computations

B. Frontend (React)

- Protocol selection and configuration UI
- Graphical node/channel representation
- Result panels for QBER, key length, and parameter logs

V. Results

Comparative protocol performance will be measured in terms of:

- Quantum Bit Error Rate (QBER)
- Sifted and secure key rate (bps)
- Impact of fiber length and noise parameters

TABLE I
SIMULATION RESULTS FOR QKD PROTOCOLS: SIFTED AND SECURE KEY RATES, QBER, AND NETWORK CONFIGURATION.

Scenario	**Distance (km)**	**Sifted Key**	**QBER (%)**	**Secure Key**	**Sifted Rate (bps)**
BB84	10	488	4.17	153	15.30 Mbps
DPS	10	115	9.09	24	2.40 Mbps
COW	10	477	6.38	126	12.60 Mbps

VI. Conclusion and Future Work

We have developed a modular simulator for BB84, DPS, and COW QKD protocols. The simulator provides students and researchers with a hands-on way to explore protocol behaviors and security principles. Future work includes:

- Adding entanglement-based protocols (e.g., E91)
- Visual QBER analysis and error correction pipelines

ACKNOWLEDGMENTS

This project is part national quantum mission of QuCIS Lab, under the supervision of Dr. Ankur Raina, IISER Bhopal. We thank our mentors and senior Sanidhya gupta for valuable feedback during development.

REFERENCES

[1] Comprehensive Analysis of BB84, A Quantum Key Distribution Protocol,arXiv:2312.05609v1 [quant-ph] 9 Dec 2023

[2] Experimental implementation of distributed phase reference quantum key distribution protocols" arXiv:2401.00146v1 [quant-ph] 30 Dec 2023

[3] D. Stucki et al., "Coherent one-way quantum key distribution," *Appl. Phys. Lett.*, vol. 87, p. 194108, 2005.

[4] Quantum Key Distribution (QKD) ,Experimental Assessment ,Overview and performance assessment of QKD system at JRC

[5] 27 Free-space quantum key distribution Alberto Carrasco-Casado; Verónica Fernández; Natalia Denisenko ,Institute of Physical and Information Technologies (ITEFI) Spanish National Research Council (CSIC) Serrano 144, Madrid 28006, Spain

[6] Field test of quantum key distribution in the Tokyo QKD Network ,23 May 2011 / Vol. 19, No. 11 / OPTICS EXPRESS 10387