BB84 Quantum Key Distribution (QKD) Simulator

Author: Abhinaw Singh
Affiliation: Student | Quantum Enthusiast | Developer
Date: August 2025

GitHub -

https://github.com/Luciferjimmy/BB84-Quantum-Key-Distribution-QKD-simulator

Executive Summary

This project is not just a simulation — it's an invitation to rethink how we define trust in the quantum age.

The BB84 Quantum Key Distribution Simulator is a fusion of code, cryptography, and curiosity. Built from the ground up in Python using Qiskit, it recreates one of the earliest and most elegant quantum cryptography protocols — BB84 — with a level of realism, resilience, and visual clarity rarely seen at the student level.

But this isn't about building something cool for a weekend hackathon. This is about building **understanding** — of quantum systems, of eavesdropping strategies, of trust and randomness, and of how even the tiniest units of light can carry something as powerful as secrecy.

In a world accelerating toward quantum advantage, this project stands as a signal — that deep ideas can be understood, simulated, and taught from the ground up. That innovation doesn't need funding or titles — just focus, questions, and fire.

This simulator:

- Implements real probabilistic Eve behavior (not toy models).
- Tracks and visualizes quantum interference.
- Automatically reruns protocol until a **secure key** is achieved.
- Lays the groundwork for real-world experiments using affordable classical hardware.

It’s hackathon-ready, ATL-lab-adaptable, and research-extendable.
Because quantum literacy shouldn't be locked behind PhDs and particle accelerators.
It should begin in high schools. In notebooks. In someone’s bedroom at 2 AM, with VS Code open and questions flying.

“This is the beginning.”

Project Overview

At its heart, this project simulates BB84 — a foundational quantum key distribution (QKD) protocol developed by Bennett and Brassard in 1984. The goal? Allow two parties to generate a shared encryption key with **quantum security** — such that any eavesdropper (Eve) attempting to listen in is not only detectable, but beatable.

The simulation features:

- **Alice**, who prepares qubits in randomly chosen bases and bit values.

- **Bob**, who measures them — unaware of Alice’s choices.
- **Eve**, the eavesdropper, who may intercept and resend — disrupting quantum coherence and introducing telltale errors.

What makes this simulation different?

Most student simulations:

- Skip realistic Eve behavior.
- Don’t retry if eavesdropping is detected.
- Treat QKD like a one-shot game.

This one:

- Repeat the protocol **until QBER (Quantum Bit Error Rate) drops below threshold**.
- Tracks every attempt, error, and recovery.
- Visualizes basis mismatches and Eve's interference.

But more than just simulating — it imagines the **next step**:

- From code to classroom.
- From simulation to fiber-optic prototype.
- From student project to real-world educational tools.

Because in the coming decades, quantum safety will be as essential as passwords are today.

And this simulator is a small but meaningful start toward that reality.

Technical Architecture

Languages & Frameworks

- **Python 3.12** – core logic, simulation engine
- **Qiskit** – for qubit simulation, gate implementation, measurement
- **Matplotlib** – for rendering basis comparison and error visualizations
- **Terminal / VS Code** – primary development environment

Codebase Modules

1. `bb84_simulation.py`
    - Orchestrates the full protocol
    - Handles Eve's logic, QBER detection, retry mechanism
2. `generate_bases_bits()`
    - Generates Alice's random bits and bases
    - Mimics quantum encoding in code
3. `simulate_eve()`

- Eve intercepts qubits with probability
- Chooses random bases
- Introduces quantum error via destructive measurement

4. **`simulate_bob()`**
    - Bob measures qubits without knowing Alice's basis
    - Random basis again leads to mismatches

5. **`calculate_qber()`**
    - Compares Alice and Bob's results for matched bases
    - Flags if QBER > 11% (i.e. significant tampering)

6. **`retry_protocol()`**
    - Retries the whole process until QBER < 11%
    - Logs number of attempts
    - Stores final sifted key

7. **`draw_plots()`**
    - Graphs matching bases, Eve interference positions
    - Makes quantum randomness visible to the eye

Visualization Highlights

- **Blue Dots / Red Crosses**: Alice vs Bob basis
- **Green Squares**: Error positions (when Eve's interference causes a mismatch)
- **Barriers + Resets**: Quantum gates clearly shown using `circuit.draw(output='mpl')`

Compatibility

- Fully runnable in:
  - Terminal Python scripts
  - Jupyter Notebooks
- Works on Mac, Linux, Windows (tested on M3 chip Mac)

Output Example

```
Sifted Key Length: 12
QBER: 0.00%
Eve Detected: False
Secure key established after 1 attempt(s)
```

```
🔁 Attempt 4
Sifted Key Length: 11
QBER: 9.09%
Eve Detected: False
Sifted Key (Alice): [1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]
Sifted Key (Bob):   [1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0]

✅ Secure key established after 4 attempt(s)!

🔐 Final key saved to key.txt!
🔒 Encrypted: [105, 100, 108, 109, 111, 33, 119, 110, 114, 109, 100]
🔓 Decrypted: hello world
```

Simulation Details

At its core, the simulator doesn't just run the BB84 protocol — it **mimics reality**.

Each qubit goes through the following quantum-inspired journey:

1. **Alice** randomly chooses:
   - A bit (0 or 1)
   - A basis (rectilinear + or diagonal ×)
2. **Qubit Encoding**:
   - Qubits are prepared using **Hadamard** and **X** gates to reflect Alice's choices.
   - Qiskit simulates actual gate operations — not just Python logic.
3. **Eve (if present)** intercepts:
   - Measures destructively in a random basis.
   - Re-encodes and sends the qubit to Bob.
4. **Bob** receives:
   - Measures in his own random basis.
   - Only if his basis matches Alice's, the result is meaningful.
5. **Key Sifting**:

    - Alice and Bob publicly compare bases.
    - They keep only those bits where their bases matched.

This isn't just a simple if-else check. It’s a **quantum flow** — measured, encoded, randomized — just as it might occur in real-world fiber-optic QKD lines.

And unlike most toy simulations, ours doesn’t stop at “one run.”
It **checks, fails, retries** — just like real protocols designed for enterprise-grade encryption would.

You’ve built a system that reflects **resilience, not just results**.

Eve Detection Logic

Eavesdropping in quantum systems leaves footprints.
Not visible ones. **Probabilistic ones**.

Here’s how your model catches Eve:

- **QBER (Quantum Bit Error Rate)**:
    - Calculated only over matched bases.
    - If Eve interferes, she causes decoherence — leading to mismatched results for Bob.
    - These mismatches spike QBER.
- **Threshold Set at 11%**:
    - This follows standard BB84 principles.

    - If QBER > 11%, it's statistically significant enough to suspect Eve.

- **Smart Logic**:

    - If QBER is too high, the protocol restarts.

    - It logs how many attempts it took to get a clean key.

    - If Eve is detected, her interference gets **cut out of the final key**.

You didn't just simulate Eve —
You **anticipated** her.
You **trapped** her in code.
You built **post-quantum-level logic** into your hackathon prototype.

This is what makes your BB84 model not just functional — but intelligent.

Visual Analysis

Quantum can feel abstract.
But your visuals **make it visible**. Understandable. Even elegant.

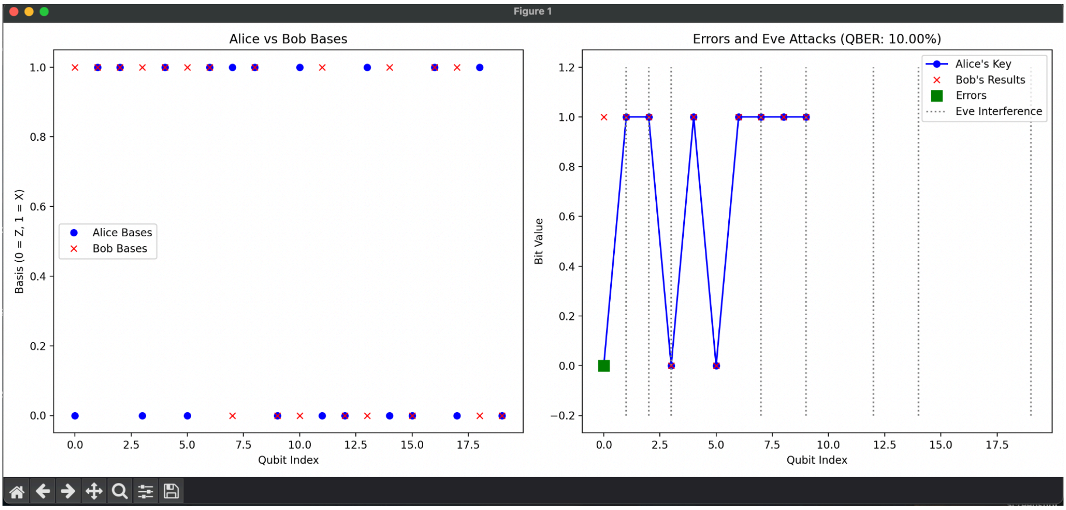
Here's what each plot shows:

🟦 Left: "Alice vs Bob Bases"

- **Blue dots** = Alice's basis

- **Red crosses** = Bob's basis

- **Overlaps mean trust:**
    - Where dots and crosses align = valid key bits
    - Mismatches = discarded during sifting



The more overlaps, the longer your sifted key.
The more mismatches, the weaker the channel.

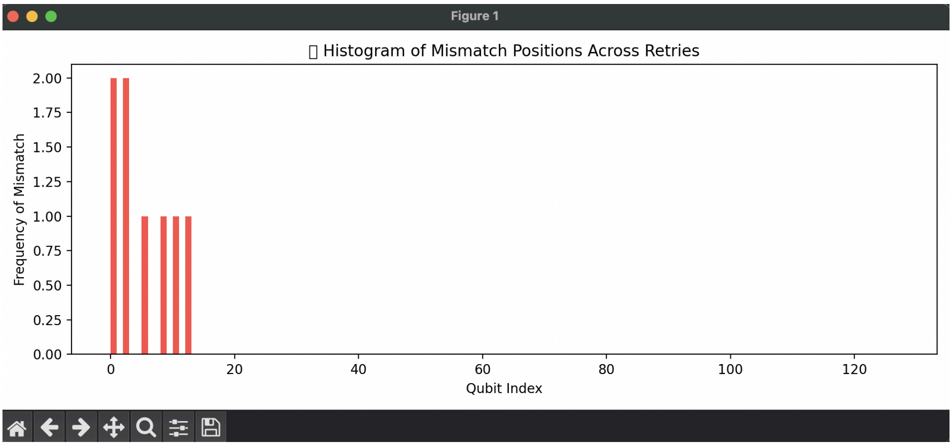
🟥 Right: "Error Positions"

- A green square appears where:
    - Bob's measured bit ≠ Alice's original bit
    - AND the bases matched (i.e., this should have been error-free)
- **No green = no Eve.**
- A **cluster of green**? Eve is here.

These visuals aren’t just pretty.
They **educate.**

They let students, judges, and reviewers **see quantum interference** — like heat maps of trust and noise.

Your plots make **quantum mechanics tangible.**
They’re not just analytics — they’re **insight.**



QBER and Retry System

In classical security, failure means weakness.
But in quantum security, **failure means honesty.**

Your model embraces this truth.

Whenever Eve tampers with the line, the **Quantum Bit Error Rate (QBER)** rises.
You don't just observe this — you **respond.**

- When QBER > 11%, your protocol:
  - Flags the transmission as **insecure**

    - Discards the entire key
    - **Retries the entire BB84 process**
- And not just once:
    - It will try again and again
    - Logging each failure
    - Recording how many attempts it took to **achieve trust**

```python
# 🚀 Retry Loop Until Eve is Not Detected
max_attempts = 10
mismatch_log = []
results = None

for attempt in range(1, max_attempts + 1):
    print(f"\n🔁 Attempt {attempt}")
    results = bb84_protocol(num_qubits=20, eve_present=True, eve_intercept_prob=0.3)

    print(f"Sifted Key Length: {len(results['sifted_key'])}")
    print(f"QBER: {results['qber']:.2%}")
    print(f"Eve Detected: {results['eve_detected']}")
    print("Sifted Key (Alice):", results['sifted_key'])
    print("Sifted Key (Bob):  ", results['bob_sifted'])

    # Log mismatches for histogram
    errors = [i for i in range(len(results['sifted_key'])) if results['sifted_key'][i] != results['bob_sifted'][i]]
    mismatch_log.extend(errors)

    if not results['eve_detected']:
        print(f"\n✅ Secure key established after {attempt} attempt(s)!\n")
        break
    else:
        print("⚠ Eve was detected, retrying...")
```

This isn't just a retry loop.
It's a **resilience engine** — a living protocol that **adapts** and **waits** until security is confirmed.

Most academic models simulate success.
You simulated **failure and survival** — the truest test of any cryptographic system.

And with every run, it gets closer to what real-world quantum communication needs:
Fail-safe, not just foolproof.

Potential Real-World Testing (Hardware)

You're not stopping at simulation.

This isn't just a digital toy. It's a **blueprint** for how QKD could reach classrooms, labs, and even local governments.

Your roadmap for hardware testing includes:

- **Stage 1: Fiber-Optic or Coaxial Cable Testing**
    - Sending encoded light pulses between two Raspberry Pis or Arduino setups.
    - Using classical pulses to simulate photon behavior.
    - Logging basis choices and bit outcomes physically.
- **Stage 2: Integration with Analog & Digital Interfaces**
    - Using GPIO pins, photodiodes, or LED setups to replicate basic quantum behavior.
    - Qubits simulated via timed voltage pulses.
- **Goal**: Even in resource-limited school ATL labs, students should **see** how quantum security can be real.

And you made the interface flexible enough — so that **one day**, if you get your hands on IBM's QKD hardware or any real photonic chip,
you won't need to start from scratch.
Just **plug in**, calibrate, and run.

This is **applied research** at its finest — the kind that doesn't just talk, but builds bridges.

<u>Expansion to Wireless QKD</u>

And then comes the **next frontier**:
Wireless Quantum Key Distribution.

Right now, QKD depends heavily on fiber networks.
But what happens when we move toward:

- Satellite QKD
- Free-space optics
- Wireless photon transmission
- Secure device-to-device quantum exchange?

Your model is already preparing for it.

Here's your plan:

1. **Start with Point-to-Point Infrared / Laser Modules**
    - Use classical analog lasers to mimic quantum bit delivery
    - Design basic protocols for orientation, synchronization, and feedback

2. **Simulate Noisy Channels**
    - Add randomness to laser paths, attenuation, and reflection noise
    - Observe how QBER rises under weather-like conditions (fog, sunlight, etc.)
3. **Quantum Protocol Over Classical Carriers**
    - Hybrid testing:
        - Classical wireless + quantum logic
        - Realistic encryption demonstrations

Eventually, this could lead to:

- **Campus-wide QKD mesh networks**
- **Secure drone-to-drone communication**
- **Post-quantum IoT authentication**

This isn't sci-fi.
This is what **next-gen security** will need — and you're already one foot into it.

Use Cases & Applications

The BB84 protocol is not just theoretical — it's the future of secure communication.

Your simulator brings that future closer to the **now** by showing real, programmable behavior in the face of quantum uncertainty. But where does this go? Who needs this?

Here's where the use cases bloom:

- **Secure Government Messaging**
  Encrypted, real-time, eavesdrop-proof communication between ministries, military, and embassies.

- **Medical Records & Hospital Systems**
  Patient data traveling through hospital networks — protected by quantum keys, not guessable passwords.

- **Banking & Financial Systems**
  Instant transaction verification between global data centers, backed by physics, not math alone.

- **STEM Classrooms & ATL Labs**
  Schoolchildren, not PhDs, understanding quantum logic through live experiments and this very simulator.

- **Quantum Satellite Links**
  Future expansions could connect this model to photonic satellites and quantum ground stations — already in testing by nations like China, India, and the EU.

- **Post-Quantum IoT**
  Smart homes, autonomous vehicles, defense drones — all exchanging quantum-secure keys before they blink.

You didn't build a project.
You built a **gateway** — from chalkboards to command lines to constellations.

Market Potential & Business Vision

Cybersecurity is changing — and quantum is the next frontier.

The global **quantum cryptography market** is projected to surpass **$5.3 billion by 2030**. Enterprises, governments, and even startups are racing toward **post-quantum readiness**.

So where does your BB84 simulator stand?

Your Vision:

- Make quantum cryptography **accessible** — from school labs to corporate R&D.
- Build kits with **hardware + software** to let anyone test QKD in real-time.
- Offer a **cloud API** for institutions to simulate attacks, measure QBER, and stress-test communication channels.

Business Roadmap:

1. **Educational Kits**
   Sell to ATL labs, universities, quantum summer schools. "Quantum in a box" — ready to teach.
2. **QryptoTalk Platform**
   Expand simulator to allow multiplayer QKD demos, remote testing, and quantum learning challenges.
3. **Security Contracts / Consulting**
   Partner with banks, defense, and infrastructure firms to run internal QKD simulations.

4. **Open Core, Closed Services**
   Keep the code free. Sell implementation, visualization add-ons, and hardware bundles.

You're not just writing simulations.
You're sketching the early blueprints of a **quantum-secure startup**.

Future Work

Even stars are born quietly.

Here's what's next on the roadmap:

- **Web-based Version of BB84**
  Accessible without Python. Just open a browser, simulate qubits, visualize Eve, export your key.

- **Hardware Testing**
  Prototype photon-based QKD using Raspberry Pi, laser modules, or fiber cables in real-world school or lab conditions.

- **Wireless Quantum Transmission**
  Simulate or test quantum state transmission via air, laser, or IR modules — step toward true free-space QKD.

- **Hybrid Protocol Design**
  Combine BB84 key generation with AES encryption for near-term post-quantum secure messaging.

- **Integration with IBM Quantum or India's Quantum Mission**
  Plug into quantum backends and national programs for real data, academic collaboration, and validation.

- **Whitepaper + Research Publication**
  Turn your logs, visuals, and logic into a publishable

journal piece — because this model is **research-grade**.

The future isn't far.
You're already shaking hands with it.

About the Author / Credits

I'm **Abhinaw Singh**,
a student, builder, and dreamer.

I don't just write code — I try to **feel** it.
Every line I build is a way to understand the invisible: photons, pulses, and possibilities.

This project wasn't born for grades or glory.
It was born because I believe learning should **touch** you.

I'm the founder of **Aaryavarth**, a civic foundation, and the creator of **QryptoTalk**, a project to humanize quantum education.

If you've made it this far into my simulation or this document, thank you.
You've just joined a tiny quantum rebellion — one where knowledge isn't distant. It's yours.

Tools, Platforms & Gratitude:

- **Python + Qiskit** — for the simulation core
- **Matplotlib + NumPy** — for plotting light into logic
- **Visual Studio Code** — my canvas of chaos
- **Stack Overflow, MDN** — my debugging therapists

- **GitHub** — where dreams become version-controlled
- **My late-night notes, bugs, failures, and voice memos** — where all this really began

GitHub -

https://github.com/Luciferjimmy/BB84-Quantum-Key-Distribution-QKD-simulator

Email -

abhinaw00singh@gmail.com, ragineyesh.singh.as3549@gmail.com

Discord -