A Simulation of the BB84 Quantum Key Distribution Protocol

Project Report

August 2, 2025

Contents

1 Introduction

1.1 Background of Quantum Cryptography

In today's digital world, securing communication is more important than ever. Classical cryptography, which relies on mathematical complexity, is facing potential threats from the rise of quantum computing. Quantum cryptography offers a new paradigm for secure communication, grounding its security in the fundamental laws of physics rather than mathematical assumptions. Unlike classical methods, it provides a way to detect eavesdropping, ensuring that the communicated information, particularly the cryptographic keys, remains confidential.

1.2 The BB84 Protocol

The BB84 protocol, developed by Charles H. Bennett and Gilles Brassard in 1984, is a pioneering method for quantum key distribution (QKD). It allows two parties, traditionally named Alice and Bob, to securely establish a shared, secret key over an insecure channel. The security of this key is guaranteed by the principles of quantum mechanics, such as the no-cloning theorem, which states that it's impossible to create an identical copy of an unknown quantum state. Any attempt by an eavesdropper, Eve, to measure the quantum states being transmitted will inevitably disturb them, introducing detectable errors that reveal her presence.

1.3 Project Objectives

The primary objective of this project is to develop a Python-based simulation of the BB84 protocol using the Qiskit library. The simulation aims to:

- **Model the key players:** Implement classes for Alice (the sender), Bob (the receiver), and Eve (the eavesdropper).
- **Simulate the protocol:** Demonstrate the complete process of key distribution, including Alice preparing and sending quantum states, Bob measuring the received states, and both parties sifting their keys to keep only the results from matching measurement bases.
- **Demonstrate eavesdropping:** Simulate an intercept-resend attack by Eve, where she intercepts the quantum states, measures them using a random basis, and sends new states to Bob based on her results.
- **Analyze the outcome:** Show how Eve's interference creates a higher Quantum Bit Error Rate (QBER), allowing Alice and Bob to detect the security breach and discard the compromised key.

1.4 Report Structure

This report is organized into several sections. Section 2 provides a theoretical overview of the BB84 protocol. Section 3 details the Python implementation, explaining the structure of the classes and key functions used in the simulation. Section 4 presents the results of the simulation, comparing a scenario without an eavesdropper to one with an active eavesdropper. Finally, Section 5 concludes with a summary of the findings and potential areas for future work.

2 The BB84 Protocol Explained

The BB84 protocol leverages principles of quantum mechanics to distribute a secret cryptographic key. Its security relies not on mathematical complexity, but on the fact that observing a quantum system fundamentally alters it.

2.1 Key Concepts

2.1.1 Quantum States and Qubits

Unlike a classical bit, which can only be a 0 or a 1, a quantum bit, or *qubit*, can exist in a superposition of both states simultaneously. The BB84 protocol uses four specific quantum states, which are grouped into two pairs known as bases:

- **Rectilinear Basis (Z-basis):** The $|0\rangle$ and $|1\rangle$ states. In the simulation, these correspond to photon polarizations of 0° and 90°, respectively.
- **Diagonal Basis (X-basis):** The $|+\rangle$ and $|-\rangle$ states. These correspond to polarizations of 45° and 135°.

2.1.2 The No-Cloning Theorem

A core principle ensuring the security of BB84 is the **no-cloning theorem**. It states that it is impossible to create an identical, independent copy of an arbitrary, unknown quantum state. This means an eavesdropper (Eve) cannot simply intercept Alice's qubit, copy it, and send the original to Bob without being detected. The simulation respects this; Eve intercepts and measures the states, she does not copy them.

2.1.3 Basis of Measurement

In quantum mechanics, the outcome of a measurement depends on the basis used.

- If a qubit is prepared in a certain basis and measured in that *same basis*, the outcome is certain. For example, measuring a $|1\rangle$ state in the Z-basis will always yield the result '1'.

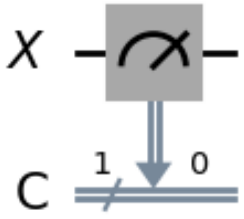

Figure 1: Standard Basis Measurement

- If it is measured in a *different basis*, the outcome becomes random. For instance, measuring a $|1\rangle$ state (Z-basis) in the X-basis has a 50% chance of resulting in '0' and a 50% chance of resulting in '1'. This property is central to detecting eavesdropping.

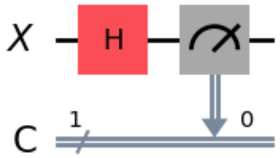

Figure 2: X Basis Measurement

2.2 Protocol Steps

The protocol is typically broken down into four distinct phases.

2.2.1 Alice's Preparation and Transmission

- **Generate Bits:** Alice begins by generating a random string of classical bits that she intends to become the secret key.
- **Generate Bases:** For each bit, she randomly chooses a measurement basis, either the Z-basis (0) or the X-basis (1).
- **Encode Qubits:** Alice encodes each classical bit as a qubit using the corresponding basis choice. For example, if her bit is '1' and her basis is 'Z', she prepares a qubit in the $|1\rangle$ state (90°). If her bit is '0' and her basis is 'X', she prepares a $|+\rangle$ state (45°).
- **Transmit:** Alice sends the sequence of prepared qubits to Bob over a quantum channel.

2.2.2 Bob's Measurement

- **Choose Bases:** For each qubit he receives from Alice, Bob independently and randomly chooses a basis (Z or X) to measure it in.
- **Measure Qubits:** Bob measures each qubit using his chosen basis and records the classical bit result (0 or 1). He is unaware of Alice's basis choices at this stage.

2.2.3 Public Discussion and Key Sifting

This phase, often called *sifting*, uses a public classical channel that is assumed to be authenticated (meaning Eve cannot tamper with the messages).

- **Compare Bases:** Bob and Alice publicly announce the sequence of bases they each used for every qubit. They do **not** reveal the bit values they measured.
- **Discard Mismatches:** They compare their basis lists and discard all the bits from their respective sequences where their chosen bases did not match. The remaining, shorter sequence of bits is known as the *sifted key*.

2.2.4 Error Rate Estimation and Eavesdropper Detection

If an eavesdropper, Eve, had intercepted the qubits, her measurements would have disturbed the states. This disturbance introduces errors in Bob's results, even when he and Alice used the same basis.

- **Compare a Subset:** Alice and Bob publicly compare a small, randomly selected portion of their sifted keys.
- **Calculate QBER:** They calculate the **Quantum Bit Error Rate (QBER)**, which is the percentage of these compared bits that do not match.
- **Detect Eve:** In an ideal, noiseless system, the QBER should be 0%. Eve's intercept-resend attack forces her to guess the basis, and when she guesses wrong, she forwards a disturbed state to Bob. This causes errors in Bob's sifted key that he and Alice can detect. If the QBER is above a certain threshold, they conclude their key has been compromised, discard it entirely, and restart the protocol.

3 Python Implementation with Qiskit

3.1 Required Libraries and Setup

This simulation is developed in Python and relies on several key libraries to handle quantum state representation, mathematical operations, and data manipulation. The primary framework used is **Qiskit**, an open-source quantum computing software development kit.

The essential libraries imported for this project are:

- **qiskit:** The core Qiskit library provides the fundamental tools for creating and working with quantum circuits and algorithms.
- **qiskit.quantum_info:** This module is crucial for the simulation. `Statevector` is used to represent and manage the quantum states of the qubits, while `Operator` is used to define quantum gates, such as the Hadamard gate (H).
- **qiskit.visualization:** Used to render quantum states in a readable format. `array_to_latex` helps in displaying quantum state vectors neatly, which is utilized by the `.draw("latex")` method on statevectors.
- **numpy:** A fundamental package for scientific computing in Python. It is used here for numerical operations, specifically to define the Hadamard operator matrix and for its `sqrt` function.
- **random:** This standard Python library is essential for simulating the non-deterministic aspects of the BB84 protocol. It's used by Alice to generate her classical bit string and basis choices, by Bob to select his measurement bases, and by Eve to decide which qubits to intercept and which bases to use for her measurement.
- **IPython.display:** The `display` function is used specifically within a Jupyter Notebook or similar environment to properly render the LaTeX output of the quantum states for clear visualization.

3.2 Core Classes and Design

The simulation is built using an object-oriented approach in Python, which organizes the complex interactions of the BB84 protocol into logical, reusable components. A parent `Person` class establishes the common attributes and behaviours, while the specialized `Alice`, `Bob`, and `Eve` classes inherit from it and implement their unique roles in the protocol.

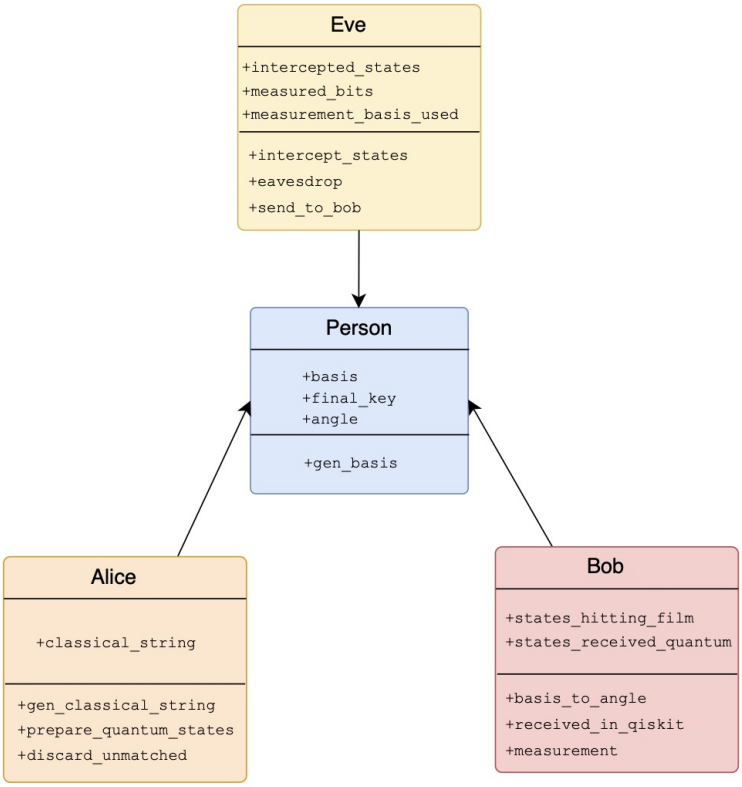

Figure 3: Class Diagram of the Core Classes

3.2.1 The Person Superclass

The `Person` class serves as a foundational blueprint for all participants in the simulation. It consolidates the attributes and methods that are common to Alice, Bob, and Eve.

Attributes:

- `basis`: Stores the basis (X or Z) used by the party.
- `final_key`: Stores the final sifted key. For a noiseless channel, Alice's and Bob's keys should match.
- `angle`: Represents the polarization angles of photons.

Methods:

- `gen_basis()`: Generates a random string of 0s (Z-basis) and 1s (X-basis).

3.2.2 The Alice Class: Key Generation and State Preparation

The `Alice` class, inheriting from `Person`, is responsible for initiating the key distribution process. She generates the initial secret information and encodes it into quantum states.

Methods and Attributes:

- `classical_string`: Stores the random classical bitstring generated by Alice which will be used to make the raw key.
- `gen_classical_string()`: Generates the classical bit string.
- `prepare_quantum_states()`: Encodes each classical bit into a polarized photon state: $0 \rightarrow 0°$, $1 \rightarrow 90°$, $+ \rightarrow 45°$, $- \rightarrow 135°$.
- `discard_unmatched()`: Discards the bits from the raw key where bases did not match to produce the sifted key.

3.2.3 The Bob Class: State Reception and Measurement

The `Bob` class, also a child of `Person`, models the recipient of the quantum states. His role is to measure the incoming qubits and collaborate with Alice to distill the final key.

Methods and Attributes:

- `states_hitting_film`: Stores the polarization angles of incoming photons.
- `received_states_quantum`: Stores the corresponding quantum state using the Qiskit library.
- `basis_to_angle()`: Converts Bob's basis choice to a measurement angle. $90°$ is used for Z-basis measurement and $135°$ for X-basis measurement.
- `received_in_qiskit()`: Converts incoming angle values into Qiskit `Statevector` objects.
- `measurement()`: For Z-basis measurement, the `measure()` function is used directly. For X-basis measurement, a Hadamard gate is applied before the standard basis measurement is performed. *[Table showing measurement outcomes to be inserted here.]*

3.2.4 The Eve Class: Simulating Eavesdropping

The `Eve` class simulates an "intercept-resend" attack to demonstrate how eavesdropping can be detected. She inherits from `Person` and implements methods to interfere with the quantum channel.

Methods and Attributes:

- `intercepted_states`: Stores the intercepted polarized states sent by Alice.
- `measured_bits`: Stores the bit values Eve obtains from her measurements.
- `measurement_basis_used`: Stores the basis Eve used for each measurement.
- `intercept_states()`: Mimics the interception of states sent by Alice. The states are stored, not measured, thus not violating the no-cloning theorem at this stage.
- `eavesdrop()`: Simulates the eavesdropping attack. Eve generates a random basis and, with 50% probability for each qubit, measures the intercepted state. If her basis matches the qubit's encoding basis, she gets a correct result; otherwise, the result is random.
- `send_to_bob()`: After measuring, Eve sends new states to Bob. Only the states she measured are modified; the rest are passed through untouched.

3.3 Key Functions

To orchestrate the interactions between the `Alice`, `Bob`, and `Eve` objects, the simulation uses several standalone helper functions. These functions control the flow of information and the sequence of events in the protocol.

3.3.1 `quantum_channel()`

This function simulates the communication channel through which Alice's qubits travel to Bob. It explicitly models Eve's intercept-resend attack. First, Eve intercepts the quantum states sent by Alice. She then executes her `eavesdrop()` method to measure a random subset of these states. Finally, she forwards a new stream of states—a mix of original and altered qubits—to Bob.

3.3.2 `matching_basis()`

This function performs the "sifting" part of the BB84 protocol. After Bob has received and measured the qubits, this function is called to compare his and Alice's basis choices. It iterates through both of their basis lists and returns a new list containing only the indices where their basis selections were identical. This list of matching indices is then used by both Alice and Bob to form their sifted keys.

3.3.3 `clearing_function()`

The `clearing_function()` is a utility designed for the simulation loop. Since the simulation runs in iterations to build a key of a target length, this function is called at the end of each round. It resets the temporary attributes of Alice, Bob, and Eve—such as their basis choices, transmitted angles, and measured bits—to empty lists, preparing them for the next iteration without affecting their final key.

3.4 Representing Quantum States and Operations

The simulation abstracts the physics of quantum mechanics by representing quantum states and measurements using classical data structures and Qiskit objects.

3.4.1 State Encoding

The encoding of classical bits into quantum states is represented by mapping bit/basis pairs to specific polarization angles. Alice performs this encoding in her `prepare_quantum_states()` method:

- **Z-basis (0):**
  - Classical bit '0' is encoded as a 0° polarization, representing the $|0\rangle$ state.
  - Classical bit '1' is encoded as a 90° polarization, representing the $|1\rangle$ state.
- **X-basis (1):**
  - Classical bit '0' is encoded as a 45° polarization, representing the $|+\rangle$ state.
  - Classical bit '1' is encoded as a 135° polarization, representing the $|-\rangle$ state.

These angles are later converted into Qiskit `Statevector` objects by Bob for measurement simulation.

3.4.2 Measurement Simulation in Qiskit

The process of quantum measurement is simulated in Bob's `measurement()` method. The simulation correctly distinguishes between measuring in the Z-basis and the X-basis:

- **Z-basis Measurement:** To measure in the standard Z-basis, the code directly calls the `.measure()` function on the `Statevector` object.
- **X-basis Measurement:** A measurement in the X-basis is equivalent to first applying a Hadamard (H) gate and then measuring in the Z-basis. The simulation achieves this by first transforming the state vector using `.evolve(H)` and then calling the `.measure()` function.

4 Simulation and Results

This section presents the findings from the Python simulation of the BB84 protocol. The results are organized into three distinct scenarios to demonstrate the protocol's functionality on a secure channel and its resilience in detecting two different types of eavesdropping attacks. The key metric for evaluating security is the Quantum Bit Error Rate (QBER).

4.1 Simulation Parameters

The simulation is configured with a specific set of parameters to demonstrate the protocol's mechanics. The simulation runs in iterations until a target key length is achieved.

- **Qubits per Iteration (`num1`)**: In each round of the simulation loop, Alice transmits a small batch of qubits to Bob.
- **Target Key Length (`total_n`)**: The simulation continues to run in iterations until a final, sifted key of a predetermined length is established between Alice and Bob.

- **Eve's Interception Rate**:We have shown the results of two experiements in case II and III : one where Eve intercepts every qubit and one where she does not. Instead, she has a **50% probability** of measuring each individual qubit that passes through the channel, as determined by the `if random.random() < 0.5:` condition in her `eavesdrop` method.

4.2 Case I: Communication without an Eavesdropper

To simulate a secure channel without an eavesdropper, a modification to the source code is required. The provided script's `quantum_channel` function hardcodes Eve's presence. To achieve a noiseless scenario, one would bypass this function and instead pass Alice's prepared states directly to Bob. This can be done by replacing the call to `quantum_channel` with the following line:

```
bob1.states_hitting_film = alice1.angle.copy()
```

The analysis below assumes this modification has been made to establish a baseline for a perfect, secure key exchange.

4.2.1 Alice's and Bob's Shared Key

In this ideal scenario where Bob receives Alice's quantum states completely unaltered, the BB84 protocol guarantees a perfectly shared secret key. After Bob measures the states and both parties publicly compare their basis choices using the `matching_basis` function, they discard all bits where their bases didn't align. The resulting sifted keys held by Alice and Bob will be **identical**.

4.2.2 Quantum Bit Error Rate (QBER) Analysis

The **Quantum Bit Error Rate (QBER)** measures the disagreements between Alice's and Bob's sifted keys. In this modified, noiseless scenario, the QBER is **0%**. A zero-error rate provides the highest confidence that the channel is secure, as no information has been disturbed or lost. The final printout would show Alice's and Bob's keys as two identical lists, confirming a successful and secure key distribution.

Alice		Bob			
State	Basis to encode (Gate)	Received	Basis	Measured	
$\|0\rangle$	Z(I)	$\|0\rangle$	Z X	0 0/1	Matching Bit Dropped
$\|1\rangle$	Z(I)	$\|1\rangle$	Z X	1 0/1	Matching Bit Dropped
$\|0\rangle$	X(H)	$\|+\rangle$	Z X	0/1 0	Dropped Matching Bit
$\|1\rangle$	X(H)	$\|-\rangle$	Z X	0/1 1	Dropped Matching Bit

Figure 4: State Analysis in Noisless case

4.3 Case II: Communication with an Eavesdropper (Inferior Approach)

This scenario analyzes an eavesdropper using a flawed ”wrong approach” strategy. The specific implementation of this attack shows that her presence is easily detectable.

4.3.1 Eve’s Interception Strategy

The simulation implements a notable variation of the ”intercept-resend” attack:

- **Intercept and Measure:** Eve intercepts all of Alice’s states and, for each one, has a 50% chance of performing a measurement. When she does, she chooses her own measurement basis (Z or X) at random.
- **Resend Z-States:** This is the most critical part of her strategy. After measuring a qubit, Eve creates and sends a new state to Bob. However, regardless of whether she measured in the Z-basis or the X-basis, she **always sends a state in the Z-basis** (0° for a ’0’ result, 90° for a ’1’ result).

This strategy is flawed from Eve’s perspective because even if she correctly guesses Alice’s basis, she will replace the original X-basis state with a Z-basis state, which will still introduce a high probability of error on Bob’s end.

4.3.2 Discrepancies in Alice’s and Bob’s Keys

Eve’s specific method of always resending Z-basis states introduces a very high number of errors. For example:

- Alice wants to send a ‘0‘ using the X-basis, so she transmits a 45° state ($|+\rangle$).

- Eve intercepts it and correctly chooses the X-basis for her measurement, getting the result '0'.
- Based on the code, Eve then sends a 0° state ($|0\rangle$) to Bob.
- Bob, also correctly choosing the X-basis for his measurement, receives the 0° state. Measuring a Z-basis state in the X-basis gives a random result.

In this case, an error is introduced even though both Eve and Bob used the correct basis. This guarantees that the final keys generated by Alice and Bob will have a large number of mismatched bits.

4.3.3 Detecting Eve's Presence through QBER

Alice and Bob can easily detect this intrusion by calculating the QBER. Given Eve's attack, the QBER will be significantly higher than zero. The theoretical QBER for this flawed approach is **37.5%** if all the qubits are intercepted and **18.75%** if there is a 50% probability of intercepting a given qubit.

Calculation Breakdown:

- **Case 1: Alice Sends a Z-basis State** (50% of the time). An error only occurs if Eve uses the wrong (X) basis. The error rate for this portion of bits is $50\% \times 50\% = \mathbf{25\%}$.
- **Case 2: Alice Sends an X-basis State** (50% of the time). Here, Eve *always* replaces it with a Z-basis state. Bob's measurement in the X-basis will therefore be random, yielding an error rate of **50%** for this portion.

The total theoretical QBER is the weighted average:
$(0.50 \times 25\%) + (0.50 \times 50\%) = 12.5\% + 25\% = \mathbf{37.5\%}$.

Experimentally, when we run our program a large number of times, the QBER rate comes closer to the theoretical value we have calculated :

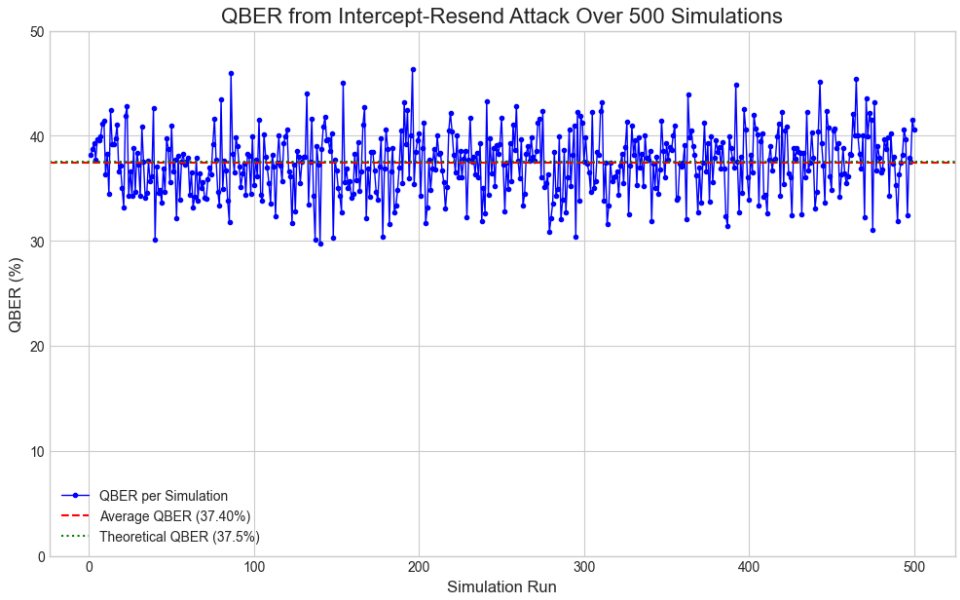

Figure 5: QBER when all qubits bits are intercepted

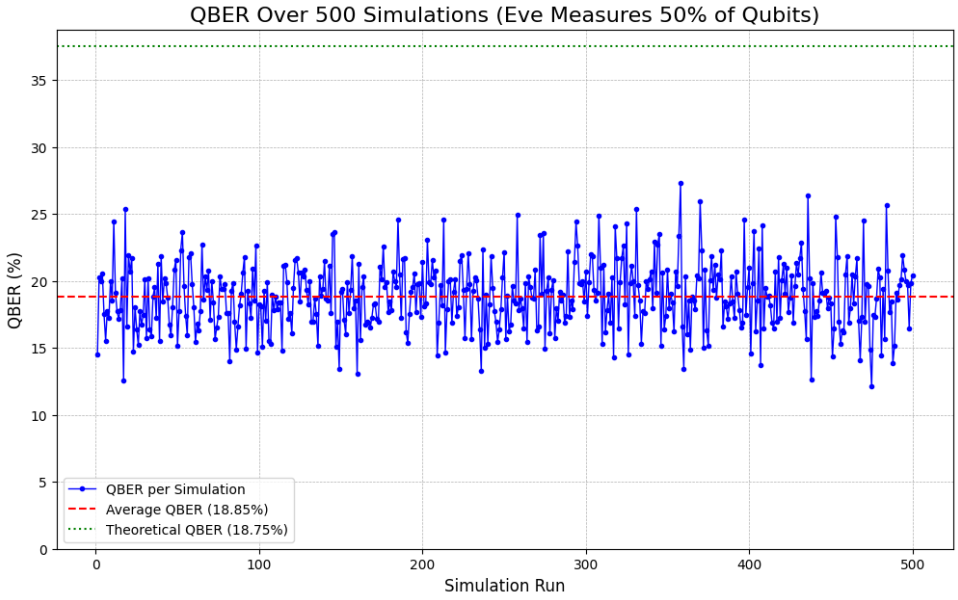

Figure 6: QBER where there is a 50% probability to intercept a qubit

4.4 Case III: Communication with an Eavesdropper (Superior strategy)

This scenario analyzes a more effective ”intercept-resend” attack. Here, Eve attempts to minimize her footprint by faithfully recreating the quantum states based on her measurements.

4.4.1 Eve’s Interception Strategy

In a more sophisticated attack, Eve’s goal is to remain as stealthy as possible by resending a quantum state that perfectly matches the result and basis of her own measurement.

- **Intercept and Measure:** Eve intercepts a qubit and randomly chooses a basis (Z or X) to measure it.
- **Resend Faithfully:** She sends a new qubit to Bob that corresponds to the basis *she used*. For example, if she measured in the X-basis and got ’1’, she sends a 135° ($|-\rangle$) state.

This is the crucial difference; by resending in the basis she measured, she avoids introducing errors when she correctly guesses Alice’s basis.

4.4.2 Discrepancies in Alice’s and Bob’s Keys

With this correct approach, errors are only introduced when Eve’s basis choice **does not match** Alice’s.

- **Correct Guess (No Error):** If Alice sends a 45° ($|+\rangle$) state and Eve also uses the X-basis, Eve will correctly measure ’0’ and resend a 45° state. Bob receives the correct state, and Eve’s presence for that bit is undetectable.
- **Incorrect Guess (Error Introduced):** If Alice sends a 45° ($|+\rangle$) state and Eve uses the Z-basis, her result is random. If she measures ’0’, she will send a 0° ($|0\rangle$) state to Bob. When Bob measures this 0° state in the correct X-basis, his result will be random, thus introducing an error.

4.4.3 Detecting Eve's Presence through QBER

This stealthier attack results in a lower, but still clearly detectable, QBER. The theoretical QBER for this attack is **25%**.

Calculation: Eve chooses the wrong basis 50% of the time. In those cases, the state she sends to Bob is incorrect. When Bob measures this incorrect state using the correct original basis, he will get the wrong answer 50% of the time. Therefore, the total error rate is 50% (Eve's wrong guess) $\times$ 50% (Bob's resulting error) $=$ **25%**. (considering all the qubits are intercepted). If there is a 50% chance of intercepting a given qubit, then OBER drops to 12.5%

While a 25% QBER is lower than that from the "wrong approach," it is still an extremely high and unambiguous signal of an eavesdropper. Upon finding an error rate anywhere near this level, Alice and Bob would immediately **discard the key** and attempt the exchange again. On a similar note, a QBER of 12.5% is also quite high and will lead to the key being discarded

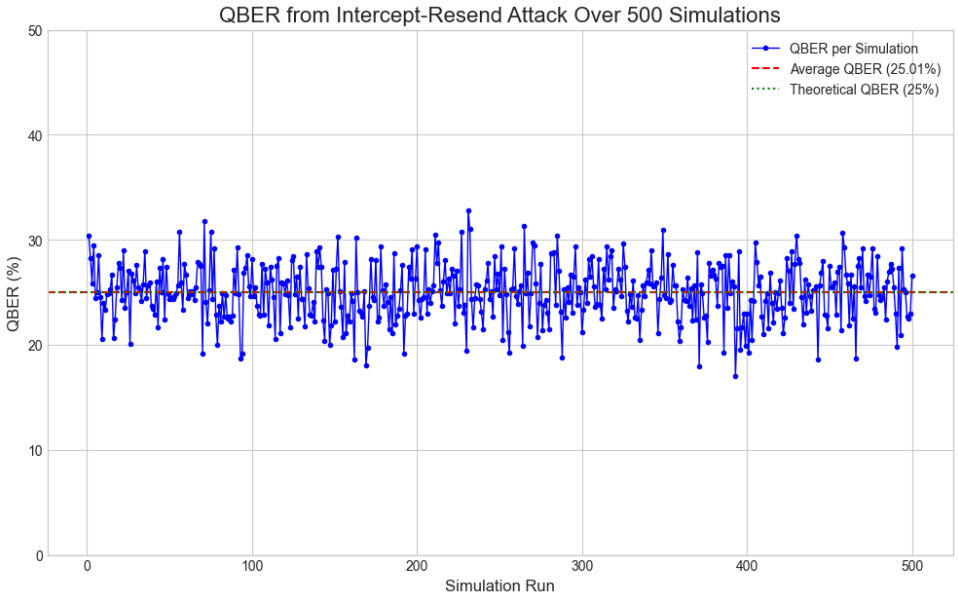

Figure 7: QBER when all qubits bits are intercepted

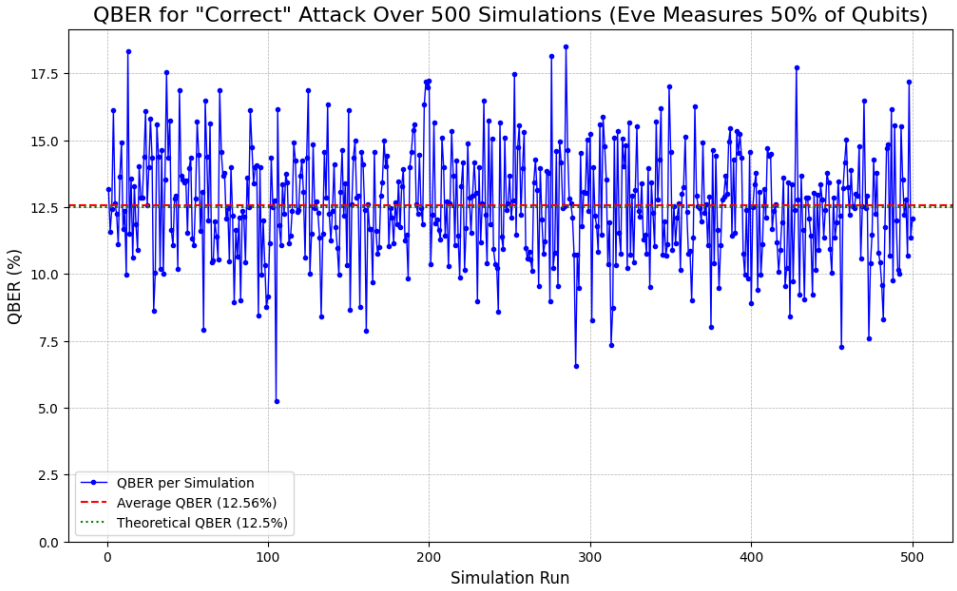

Figure 8: QBER where there is a 50% probability to intercept a qubit

5 Conclusion

This project successfully demonstrates the principles of the BB84 quantum key distribution protocol through a Python simulation. By modeling the roles of Alice, Bob, and an eavesdropper, Eve, the simulation provides a practical understanding of how quantum mechanics ensures secure communication.

5.1 Summary of Findings

The simulation effectively illustrates the core concepts of the BB84 protocol. In a secure, noiseless scenario, Alice and Bob can establish a perfectly identical secret key. The introduction of an eavesdropper, Eve, invariably disturbs the quantum states, leading to a detectable Quantum Bit Error Rate (QBER). We found that a flawed eavesdropping strategy results in a high QBER of 37.5%, while a more sophisticated intercept-resend attack produces a theoretical QBER of 25%. In both cases, the error rate is significant enough to alert Alice and Bob of the security breach, confirming that the protocol's security is guaranteed by the laws of physics.

5.2 Limitations of the Simulation

The simulation effectively illustrates the core concepts of the BB84 protocol. In a secure, noiseless scenario, Alice and Bob can establish a perfectly identical secret key. The introduction of an eavesdropper, Eve, invariably disturbs the quantum states, leading to a detectable Quantum Bit Error Rate (QBER). We found that a flawed eavesdropping strategy results in a high QBER of 37.5%, while a more sophisticated intercept-resend attack produces a theoretical QBER of 25%. In both cases, the error rate is significant enough to alert Alice and Bob of the security breach, confirming that the protocol's security is guaranteed by the laws of physics.

5.3 Future Work and Potential Improvements

Based on the current limitations, several improvements could be made in future work to enhance the simulation's realism and scope:

- **Implement a Noise Model:** Introduce a variable to simulate environmental noise, allowing for the analysis of how a baseline QBER affects the threshold for detecting an eavesdropper.
- **Add Post-Processing Algorithms:** Implement classical algorithms for error correction (like a simplified parity check or the Cascade protocol) and privacy amplification (using a hash function) to simulate the full QKD process.
- **Explore Other Protocols and Attacks:** The framework could be extended to simulate other QKD protocols, such as E91 or the six-state protocol, or to model more advanced eavesdropping strategies.

6 References

References

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

[2] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[3] N. Gisin, G. Ribordy, W. Tittel, and H. Zbinden, "Quantum cryptography," *Reviews of Modern Physics*, vol. 74, no. 1, pp. 145–195, 2002.

[4] Qiskit Development Team, *Qiskit Textbook*. IBM, 2024. [Online]. Available: https://qiskit.org/textbook/

7 Appendices

7.1 Full Python Code

Code 1

```
from qiskit import __version__
import numpy as np
from qiskit.visualization import array_to_latex
from qiskit.quantum_info import Statevector
from qiskit.visualization import plot_histogram
from qiskit.quantum_info import Operator
from qiskit import QuantumCircuit
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit.quantum_info import Operator
from qiskit_aer import AerSimulator
from qiskit.result import marginal_distribution
from qiskit.circuit.library import UGate
```

```
import random
from numpy import sqrt, pi, random
from IPython.display import display

# Define helper functions
def quantum_channel(alice, bob):
  bob.states_hitting_film = alice.angle

def matching_basis(alice, bob):
  matched_basis_index = []
  for i, (a, b) in enumerate(zip(alice.basis, bob.basis)):
  if a == b:
    matched_basis_index.append(i)
    return matched_basis_index

def clearing_function(alice, bob):
  alice.basis = []
  alice.angle = []
  alice.classical_string = []
  bob.basis = []
  bob.angle = []
  bob.states_hitting_film = []
  bob.received_states_quantum = []

# Define main classes
class Person():
  def __init__(self):
    self.basis = []
    self.final_key = []
    self.angle = []

def gen_basis(self, num):
  self.basis = [random.randint(0, 2) for _ in range(num)]

class Alice(Person):
  def __init__(self):
    super().__init__()
    self.classical_string = []

def gen_classical_string(self, num):
  self.classical_string = [random.randint(0, 2) for _ in range(num)]

def prepare_quantum_states(self):
  self.angle = []
  for i, j in zip(self.classical_string, self.basis):
    if i == 0 and j == 0:
      self.angle.append(0)
    elif i == 1 and j == 0:
      self.angle.append(90)
    elif i == 0 and j == 1:
      self.angle.append(45)
    elif i == 1 and j == 1:
      self.angle.append(135)

def discard_unmatched(self, matched_basis_index):
  for i in matched_basis_index:
    self.final_key.append(self.classical_string[i])

```

```
class Bob(Person):
  def __init__(self):
    super().__init__()
    self.states_hitting_film = []
    self.received_states_quantum = []

def basis_to_angle(self):
  self.angle = []
  for b in self.basis:
    if b == 1:
      self.angle.append(135)
    else:
      self.angle.append(90)

def received_in_qiskit(self):
  self.received_states_quantum = []
  for angle in self.states_hitting_film:
  if angle == 0:
    self.received_states_quantum.append(Statevector.from_label("0"))
  elif angle == 90:
    self.received_states_quantum.append(Statevector.from_label("1"))
  elif angle == 45:
    self.received_states_quantum.append(Statevector.from_label("+"))
  elif angle == 135:
    self.received_states_quantum.append(Statevector.from_label("-"))

def measurement(self, matched_basis_index):
  for i in matched_basis_index:
    if self.angle[i] == 90:
      bit, _ = self.received_states_quantum[i].measure([0])
      self.final_key.append(int(bit))
    elif self.angle[i] == 135 and self.states_hitting_film[i] == 135:
      self.final_key.append(1)
    elif self.angle[i] == 135 and self.states_hitting_film[i] == 45:
      self.final_key.append(0)

# Run simulation
total_n = 16
alice1 = Alice()
bob1 = Bob()
count = 1

while len(bob1.final_key) < total_n:
  print("----------Iteration ", count, '------------------')

  num1 = 4
  alice1.gen_classical_string(num1)
  print('Classical String generated by Alice:', alice1.classical_string)

  alice1.gen_basis(num1)
  print('Basis chosen by Alice:', alice1.basis)

  alice1.prepare_quantum_states()
  print("Quantum states sent by Alice as polarized light:", alice1.angle)

  quantum_channel(alice1, bob1)

  bob1.gen_basis(num1)
```

```
    print('Basis chosen by Bob:', bob1.basis)

    bob1.basis_to_angle()
    print("Angle of polarized films used by Bob:", bob1.angle)
    bob1.received_in_qiskit()
    for state in bob1.received_states_quantum:
    display(state.draw("latex"))
    match = matching_basis(alice1, bob1)
    print("Basis matched at:", match)
    alice1.discard_unmatched(match)
    bob1.measurement(match)
    print("Alice's final key so far:", alice1.final_key)
    print("Bob's final key so far:  ", bob1.final_key)
    clearing_function(alice1, bob1)
    count += 1

```

Listing 1: Full Python script for the BB84 simulation without an eavesdropper.

Code 2

```
import numpy as np
from qiskit.visualization import array_to_latex
from qiskit.quantum_info import Statevector, Operator
from qiskit.visualization import plot_histogram
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit_aer import AerSimulator
from qiskit.result import marginal_distribution
from qiskit.circuit.library import UGate
import random
from numpy import sqrt, pi, random
from IPython.display import display

H = Operator([[1 / sqrt(2), 1 / sqrt(2)], [1 / sqrt(2), -1 / sqrt(2)]])

# Helper functions
def quantum_channel(alice, eve, bob):
  eve.intercept_states(alice.angle)
  print("\nEve's Interception:")
  print(f"States received from Alice: {[f'{x}°' for x in eve.
 intercepted_states]}")
  eve.eavesdrop()
  bob.states_hitting_film = eve.send_to_bob()
  print(f"States sent to Bob: {[f'{x}°' for x in bob.states_hitting_film
 ]}")

def matching_basis(alice, bob):
  return [i for i, (a, b) in enumerate(zip(alice.basis, bob.basis)) if a
 == b]

def clearing_function(alice, eve, bob):
  alice.basis = []
  alice.angle = []
  alice.classical_string = []
  bob.basis = []
  bob.angle = []
  bob.states_hitting_film = []
  bob.received_states_quantum = []
  eve.basis = []
  eve.measured_bits = []
  eve.intercepted_states = []

# Main classes
class Person():
  def __init__(self):
    self.basis = []
    self.final_key = []
    self.angle = []

def gen_basis(self, num):
  self.basis = [random.randint(0, 2) for _ in range(num)]

class Alice(Person):
  def __init__(self):
    super().__init__()
    self.classical_string = []

def gen_classical_string(self, num):
```

```
    self.classical_string = [random.randint(0, 2) for _ in range(num)]

  def prepare_quantum_states(self):
    self.angle = []
    for i, j in zip(self.classical_string, self.basis):
      if i == 0 and j == 0:
        self.angle.append(0)
      elif i == 1 and j == 0:
        self.angle.append(90)
      elif i == 0 and j == 1:
        self.angle.append(45)
      elif i == 1 and j == 1:
        self.angle.append(135)

  def discard_unmatched(self, matched_basis_index):
    for i in matched_basis_index:
    self.final_key.append(self.classical_string[i])

  class Eve(Person):
    def __init__(self):
      super().__init__()
      self.intercepted_states = []
      self.measured_bits = []
      self.measurement_basis_used = []

  def intercept_states(self, states):
    self.intercepted_states = states.copy()

  def eavesdrop(self):
    num_bits = len(self.intercepted_states)
    self.gen_basis(num_bits)
    self.measured_bits = []
    self.measurement_basis_used = []

    for i in range(num_bits):
      if random.random() < 0.5:
        state = self.intercepted_states[i]
        basis_used = self.basis[i]
        self.measurement_basis_used.append((i, basis_used))

        if basis_used == 0:  # Z-basis
          if state in [0, 90]:
            measured_bit = int(state / 90)
          else:
            measured_bit = random.randint(0, 2)
            self.measured_bits.append((i, measured_bit))

        elif basis_used == 1:  # X-basis
          if state == 45:
            measured_bit = 0
          elif state == 135:
            measured_bit = 1
          else:
            measured_bit = random.randint(0, 2)
          self.measured_bits.append((i, measured_bit))

    print("\nEve's Measurement Details:")
    if not self.measurement_basis_used:
```

```
      print("Eve didn't measure any bits this round")
    else:
      for i, basis in self.measurement_basis_used:
        original_state = self.intercepted_states[i]
        measured_bit = next((bit for idx, bit in self.measured_bits if idx
  == i), None)
        print(f"Bit {i}: Original {original_state}° - Measured in {'X' if
  basis else 'Z'}-basis - Got {measured_bit}")

  def send_to_bob(self):
    new_states = self.intercepted_states.copy()
    for i, bit_val in self.measured_bits:
    # Check the basis Eve used for the i-th qubit
      if self.basis[i] == 0: # Resend in Z-basis
        new_states[i] = 0 if bit_val == 0 else 90
      else: # Resend in X-basis
        new_states[i] = 45 if bit_val == 0 else 135
    return new_states

  class Bob(Person):
    def __init__(self):
      super().__init__()
      self.states_hitting_film = []
      self.received_states_quantum = []

  def basis_to_angle(self):
    self.angle = [135 if b == 1 else 90 for b in self.basis]

  def received_in_qiskit(self):
    self.received_states_quantum = []
    for angle in self.states_hitting_film:
      if angle == 0:
        self.received_states_quantum.append(Statevector.from_label("0"))
      elif angle == 90:
        self.received_states_quantum.append(Statevector.from_label("1"))
      elif angle == 45:
        self.received_states_quantum.append(Statevector.from_label("+"))
      elif angle == 135:
        self.received_states_quantum.append(Statevector.from_label("-"))

  def measurement(self, matched_basis_index):
    for i in matched_basis_index:
    if self.angle[i] == 90:
      bit, _ = self.received_states_quantum[i].measure([0])
      print(f"Bit at position {i} is : {bit}")
      self.final_key.append(int(bit))
    elif self.angle[i] == 135:
      self.received_states_quantum[i] = self.received_states_quantum[i].
  evolve(H)
      bit, _ = self.received_states_quantum[i].measure([0])
      print(f"Bit at position {i} is : {bit}")
      self.final_key.append(int(bit))

  # Run simulation
  total_n = 128
  alice1 = Alice()
  eve = Eve()
  bob1 = Bob()
```

```
count = 1

while len(bob1.final_key) < total_n:
  print("\n" + "="*50)
  print(f"---------- Iteration {count} ------------------")
  print("="*50)

  num1 = 24
  alice1.gen_classical_string(num1)
  print('\nAlice:')
  print(f'Classical String: {alice1.classical_string}')
  alice1.gen_basis(num1)
  print(f'Basis chosen: {["Z" if b == 0 else "X" for b in alice1.basis]}'
 )
  alice1.prepare_quantum_states()
  print(f"Quantum states: {[f'{x}°' for x in alice1.angle]}")

  quantum_channel(alice1, eve, bob1)

  bob1.gen_basis(num1)
  print('\nBob:')
  print(f'Basis chosen: {["Z" if b == 0 else "X" for b in bob1.basis]}')
  bob1.basis_to_angle()
  print(f"Measurement angles: {[f'{x}°' for x in bob1.angle]}")

  bob1.received_in_qiskit()
  print("\nBob's received states:")
  for i, state in enumerate(bob1.received_states_quantum):
  print(f"Bit {i}: ", end="")
  display(state.draw("latex"))

  match = matching_basis(alice1, bob1)
  print(f"\nBasis matched at indices: {match}")

  alice1.discard_unmatched(match)
  bob1.measurement(match)
  print(f"\nAlice's final key so far: {alice1.final_key}")
  print(f"Bob's final key so far:   {bob1.final_key}")
  print(f"Current key length: {len(bob1.final_key)}/{total_n}")

  clearing_function(alice1, eve, bob1)
  count += 1

print("\n" + "="*50)
print("Final Results:")
print("="*50)
print(f"Alice's final key: {alice1.final_key}")
print(f"Bob's final key:   {bob1.final_key}")
print(f"Key length: {len(bob1.final_key)} bits")
print(f"Number of iterations: {count-1}")
```

Listing 2: Full Python script for the BB84 simulation with an eavesdropper where each qubit has a 50% chance of being intercepted (superior approach).

Code 3

```
  def send_to_bob(self):
    new_states = self.intercepted_states.copy()
    for i, bit in self.measured_bits:
      new_states[i] = 0 if bit == 0 else 90
      return new_states
```

Listing 3: Snippet for Inferior Approach).

Code 4

```
def eavesdrop(self):
  # Eve measures every qubit to maximize her information
  num_bits = len(self.intercepted_states)
  self.gen_basis(num_bits)
  self.measured_bits = []

  for i in range(num_bits):
    state = self.intercepted_states[i]
    basis_used = self.basis[i]
    measured_bit = 0

    if basis_used == 0:  # Z-basis measurement
      if state in [0, 90]: measured_bit = state // 90
      else: measured_bit = random.randint(0, 1) # Guess for X-basis states
    else:  # X-basis measurement
      if state in [45, 135]: measured_bit = (state - 45) // 90
      else: measured_bit = random.randint(0, 1) # Guess for Z-basis states
    self.measured_bits.append(measured_bit)
```

Listing 4: Snippet for full interception).