BB84 Practical Implementation

November 14, 2025

1 BB84 Simulation with Eve (Intercept-Resend Attack)

This notebook simulates 20 rounds of the BB84 protocol with an eavesdropper, Eve, performing an intercept-and-resend attack.

1.0.1 1. Imports and Setup

First, we import `cirq` for quantum simulation, `numpy` for random choices, and `pandas` to display the results in a clean table. We also define our two bases: 'R' (Rectilinear/Z-basis) and 'D' (Diagonal/X-basis).

[1]:
```
# Install necessary libraries
!pip install -q cirq pandas
import cirq
import numpy as np
import pandas as pd

# Define the two bases: Z (Rectilinear, R) and X (Diagonal, D)
BASIS_Z = 'R' # Z-basis, Rectilinear
BASIS_X = 'D' # X-basis, Diagonal
```

1.0.2 2. Helper Functions

We need functions to simulate the actions of Alice, Eve, and Bob.

- `prepare_qubit`: Simulates preparing a qubit in one of the four BB84 states ($|0\rangle, |1\rangle, |+\rangle, |-\rangle$). This is used by Alice to create the initial qubit and by Eve to create her forged qubit.
- `measure_qubit`: Simulates measuring a qubit in either the 'R' (Z) or 'D' (X) basis. This is used by Eve to intercept the qubit and by Bob to get his final result.

[2]:
```
def prepare_qubit(bit, basis):
    """Alice or Eve prepares a qubit based on their bit and basis choice."""
    qubit = cirq.LineQubit(0)

    if bit == 0:
        # Prepare |0> (for Z) or |+> (for X)
        if basis == BASIS_Z:
            # |0> state, no operation needed
            return qubit, []
        else:
            # |+> state
            return qubit, [cirq.H(qubit)]
    else:
        # Prepare |1> (for Z) or |-> (for X)
        if basis == BASIS_Z:
            # |1> state
            return qubit, [cirq.X(qubit)]
        else:
            # |-> state
            return qubit, [cirq.X(qubit), cirq.H(qubit)]

def measure_qubit(qubit, circuit, basis):
    """Bob or Eve measures the qubit in their chosen basis."""

    if basis == BASIS_Z:
        # Z-basis measurement is standard
        circuit.append(cirq.measure(qubit, key='result'))
    else:
        # X-basis measurement (apply Hadamard first)
        circuit.append(cirq.H(qubit))
        circuit.append(cirq.measure(qubit, key='result'))

    # Simulate the circuit
    simulator = cirq.Simulator()
    result = simulator.run(circuit, repetitions=1)
    measurement = result.measurements['result'][0][0]
    return measurement
```

1.0.3 3. Single Round Simulation

This function combines the preparation and measurement steps to simulate a single, full round of the protocol, from Alice to Eve to Bob.

1. **Alice** randomly chooses a bit and a basis, then prepares a qubit.
2. **Eve** randomly chooses a basis, measures Alice's qubit, and gets a bit.
3. **Eve** then prepares a *new* qubit based on the bit and basis she just used.
4. **Bob** randomly chooses a basis and measures the new qubit from Eve.

[3]:
```
def run_bb84_round():
    """Simulates one full round of BB84 with Alice, Eve, and Bob."""

    # 1. ALICE
    alice_basis = np.random.choice([BASIS_Z, BASIS_X])
    alice_bit = np.random.choice([0, 1])
    qubit, prep_ops = prepare_qubit(alice_bit, alice_basis)
    alice_circuit = cirq.Circuit(prep_ops)
    # 2. EVE (Intercept-Resend Attack)
    eve_basis = np.random.choice([BASIS_Z, BASIS_X])
    # Eve measures Alice's qubit
    eve_circuit = alice_circuit.copy()
    eve_bit = measure_qubit(qubit, eve_circuit, eve_basis)

    # Eve prepares a *new* qubit to send to Bob
    eve_qubit_to_bob, eve_prep_ops = prepare_qubit(eve_bit, eve_basis)

    # 3. BOB
    bob_basis = np.random.choice([BASIS_Z, BASIS_X])
    bob_circuit = cirq.Circuit(eve_prep_ops)
    bob_bit = measure_qubit(eve_qubit_to_bob, bob_circuit, bob_basis)

    return {
        "Alice Basis": alice_basis,
        "Alice Bit": alice_bit,
        "Eve Basis": eve_basis,
        "Eve Bit": eve_bit,
        "Bob Basis": bob_basis,
        "Bob Bit": bob_bit
    }
```

1.0.4 4. Run the 20-Round Simulation

Now we run the simulation 20 times and store the results in a pandas DataFrame to view them clearly.

[4]:
```
print("--- Running 20-Round BB84 Simulation (with Eve) ---")

results = []
for i in range(20):
    results.append(run_bb84_round())

# Display results in a clean table
df = pd.DataFrame(results)
df.index.name = "Round"
print(df.to_string())
```

```
--- Running 20-Round BB84 Simulation (with Eve) ---
      Alice Basis  Alice Bit Eve Basis  Eve Bit Bob Basis  Bob Bit
Round
0                D          0         R        1         R        1
1                R          0         D        1         D        1
2                D          0         D        0         R        0
3                R          1         D        0         R        0
4                D          1         D        1         R        1
5                R          1         R        1         D        0
```

```
6           D        0        D        0        R        0
7           R        1        R        1        D        1
8           D        0        D        0        D        0
9           R        1        R        1        R        1
10          D        0        R        0        D        0
11          D        1        R        0        R        0
12          R        1        D        0        R        1
13          R        1        D        1        D        1
14          D        0        R        1        R        1
15          R        0        D        0        D        0
16          D        1        D        1        R        1
17          D        0        R        0        R        0
18          R        1        R        1        R        1
19          R        0        R        0        D        1
```

1.0.5 5. Sifting and QBER Calculation

This is the final step, corresponding to **Questions 2.6 and 2.8**.

1. **Sifting:** We simulate the public channel discussion by keeping *only* the rounds where Alice and Bob's basis choices matched ('R'=='R' or 'D'=='D').
2. **QBER Calculation:** We compare Alice's original bits and Bob's measured bits *in the sifted rounds* to find the error rate.

[5]:
```python
print("\n" + "="*70 + "\n")
print("--- Sifting and QBER Calculation ---")

# Sifting: Keep only rounds where Alice and Bob's bases match
sifted_df = df[df["Alice Basis"] == df["Bob Basis"]].copy()

if len(sifted_df) == 0:
    print("No rounds had matching bases! (Unlikely, try running again)")
else:
    print(f"Bases matched for {len(sifted_df)} out of 20 rounds.")

    # Compare Alice's and Bob's bits in the sifted rounds
    sifted_df["Error"] = (sifted_df["Alice Bit"] != sifted_df["Bob Bit"])

    alice_sifted_key = "".join(sifted_df["Alice Bit"].astype(str))
    bob_sifted_key = "".join(sifted_df["Bob Bit"].astype(str))

    print(f"\nAlice's Sifted Key: {alice_sifted_key}")
    print(f"Bob's Sifted Key:   {bob_sifted_key}")

    # Calculate QBER
    num_errors = sifted_df["Error"].sum()
    num_sifted_bits = len(sifted_df)
    # Avoid division by zero if no bits were sifted
    if num_sifted_bits > 0:
        qber = num_errors / num_sifted_bits
    else:
        qber = 0 # Or float('nan')

    print("\n--- Final Result (for Q2.8) ---")
    print(f"Total Sifted Bits: {num_sifted_bits}")
    print(f"Total Errors Found: {num_errors}")

    if num_sifted_bits > 0:
        print(f"QBER = {num_errors} / {num_sifted_bits} = {qber:.2%}")
    else:
        print("QBER = N/A (no sifted bits)")

    print("\nThis QBER is the result of Eve's intercept-resend attack.")
    print("The theoretical expected QBER is 25%. Your simulation result should␣
↪be close to this.")
```

```
======================================================================

--- Sifting and QBER Calculation ---
Bases matched for 6 out of 20 rounds.

Alice's Sifted Key: 101011
Bob's Sifted Key:   001011

--- Final Result (for Q2.8) ---
Total Sifted Bits: 6
Total Errors Found: 1
QBER = 1 / 6 = 16.67%

This QBER is the result of Eve's intercept-resend attack.
The theoretical expected QBER is 25%. Your simulation result should be close to
this.
```

1.0.6 Conclusion

This practical session successfully demonstrated the BB84 protocol.

- **Theory:** We mathematically confirmed that an intercept-resend attack introduces a **25% QBER**, which results in a **negative key rate**, forcing the protocol to abort.
- **Practice (Key Exchange):** We performed a 20-round key exchange (Q2.5) and successfully sifted a **7-bit key** (0110010) with an ideal **QBER of 0%** (Q2.6, Q2.8). This demonstrates the protocol's correctness in an error-free environment.
- **Practice (Eavesdropping):** The experimental eavesdropping section (Q2.7) was replaced with a Cirq simulation. The simulation (results attached) produced a **16.67% QBER**. This non-zero result confirms that Eve's presence introduces detectable errors, validating the secu-

rity principle of the protocol. #

- **Final Insight:** The experiment highlights the fundamental difference between this classical-analog (which is insecure to beam-splitting) and a true quantum system, which relies on single photons and the no-cloning theorem to be secure.