# BB84 Quantum Key Distribution Experiment Report

**Author:** Tymon Dydowicz
**Environment:** Qiskit, Python 3.10.11
**Student ID:** 151936

## Project Overview

This project implements and evaluates a simplified **BB84 Quantum Key Distribution (QKD)** protocol using Qiskit.
The experiment examines how the **final sifted key length** scales with the **number of qubits transmitted**, averaged over 5 executions for each sample size.

## Experiment Parameters

**Simulator** Qiskit AerSimulator
**Qubits Used** 4
**Samples Tested** 16, 32, 64, 128, 256, 512, 1024
**Executions per Sample** 5
**Random Seed** 151936
**Measurement Shots** 1 per circuit run

## Important Code

```
def createBB84Circuit():
    q0 = QuantumRegister(N_QUBITS, 'q')
    c0 = ClassicalRegister(N_QUBITS, 'c')

    all_qubits = [q0[i] for i in range(N_QUBITS)]

    circ = QuantumCircuit(q0, c0)
    circ.reset(all_qubits)
    circ.h(q0[1])
    circ.measure(q0[1], c0[1])
    circ.h(q0[2])
    circ.measure(q0[2], c0[2])
    circ.barrier(all_qubits)

    circ.cx(q0[1], q0[0])
    circ.ch(q0[2], q0[0])
    circ.barrier(all_qubits)

    circ.h(q0[3])
    circ.measure(q0[3], c0[3])
    circ.barrier(all_qubits)
```

```
    circ.ch(q0[3], q0[0])
    circ.measure(q0[0], c0[0])
    circ.barrier(all_qubits)

    return circ
```

```
def testSample(circ, sample, verbose=False):
    bits = []
    for _ in range(sample):
        compiled_circuit = transpile(circ, BACKEND)
        job_sim = BACKEND.run(compiled_circuit, shots=1)
        sim_result = job_sim.result()
        counts = sim_result.get_counts(circ)

        xA = int(list(counts.keys())[0][2])
        yA = int(list(counts.keys())[0][1])
        yB = int(list(counts.keys())[0][0])
        xB = int(list(counts.keys())[0][3])

        if verbose:
            print(f"Alice bits: xA={xA}, yA={yA} | Bob bits: yB={yB}, xB={xB}")
            print([xA, yA, yB, xB])
        bits.append([xA, yA, yB, xB])

    return bits
```

```
def siftKey(bits):
    keyA = []
    keyB = []
    for bit in bits:
        xA, yA, yB, xB = bit
        if yA == yB:
            keyA.append(xA)
            keyB.append(xB)
    return keyA, keyB
```

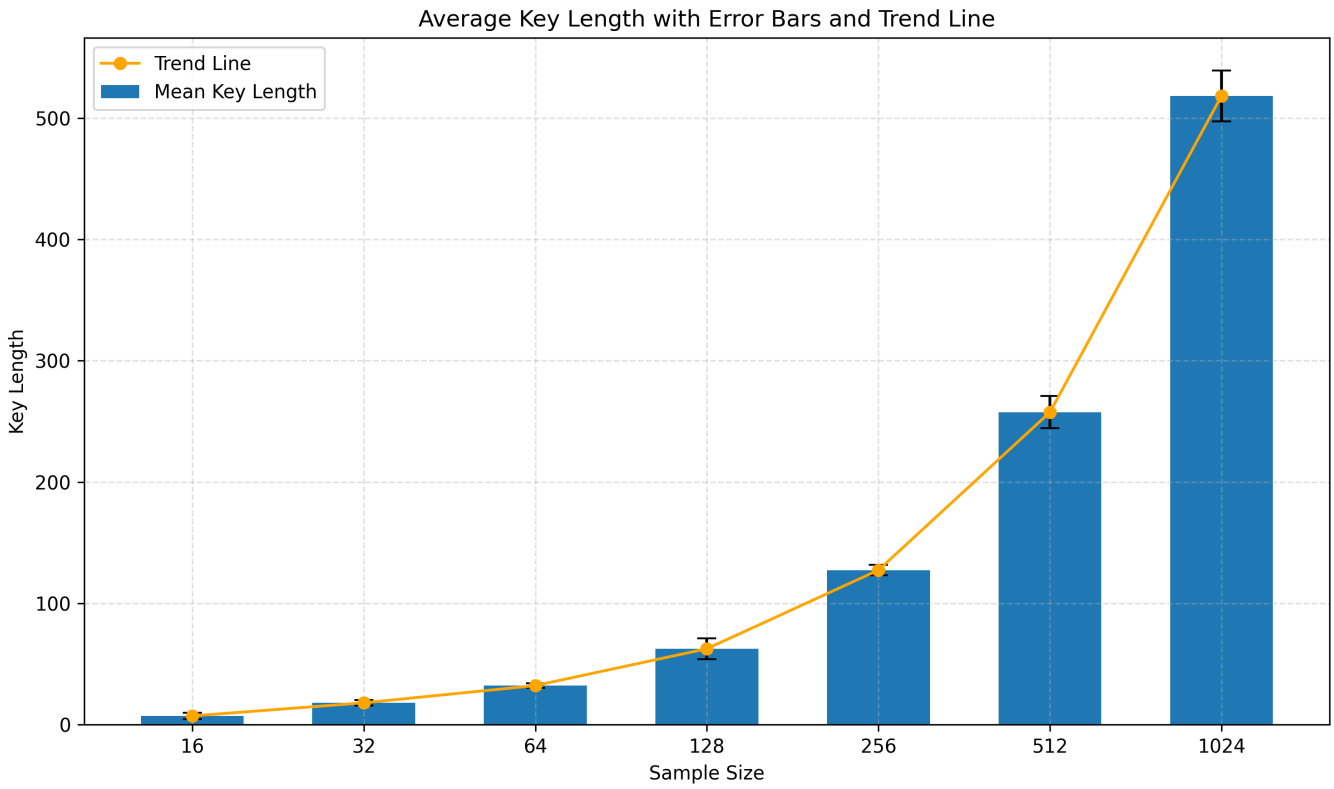
```
def runExperiment(sample, verbose=False):
    circ = createBB84Circuit()
    bits = testSample(circ, sample)
    keyA, keyB = siftKey(bits)
    if verbose:
        print(f"Alice's key: {keyA}")
        print(f"Bob's key:   {keyB}")
    return keyA, keyB
```

# Key Length Results Table

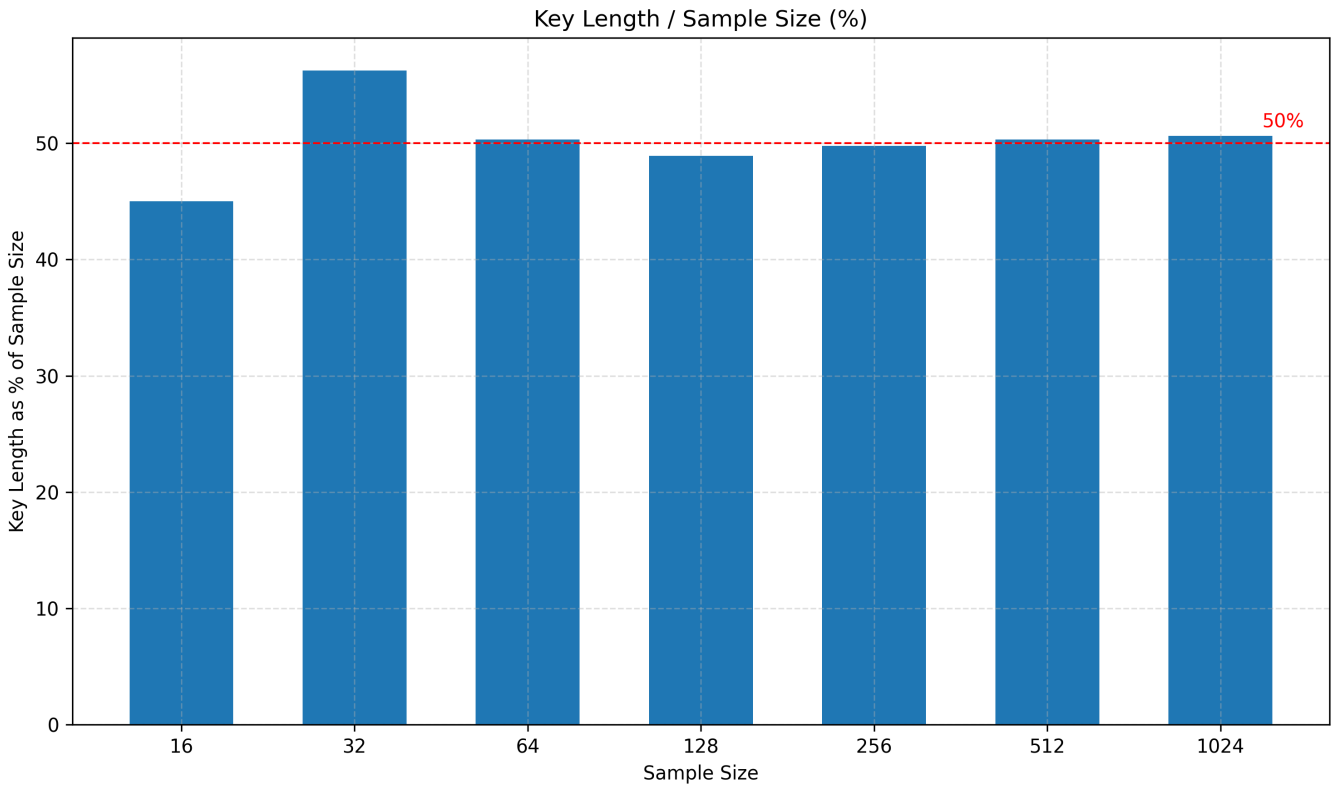
| Sample Size | 1 | 2 | 3 | 4 | 5 | avg | std |
|---|---|---|---|---|---|---|---|
| 16 | 9 | 4 | 9 | 9 | 5 | 7.20 | 2.49 |
| 32 | 18 | 15 | 21 | 19 | 17 | 18.00 | 2.24 |
| 64 | 30 | 31 | 34 | 34 | 32 | 32.20 | 1.79 |
| 128 | 65 | 62 | 69 | 69 | 48 | 62.60 | 8.68 |
| 256 | 127 | 129 | 121 | 133 | 127 | 127.40 | 4.34 |
| 512 | 256 | 259 | 236 | 268 | 269 | 257.60 | 13.31 |
| 1024 | 487 | 518 | 513 | 541 | 533 | 518.40 | 20.85 |

# Visualizations

### 1. Average Key Length With Error Bars and Trend Line




### 2. Key Length as % of Sample Size




---

# Interpretation

## Scaling Behavior

- The sifted key length grows **approximately linearly** with sample size.
- The percent of usable key stabilizes around **50%**, matching the theoretical expectation for BB84 with random bases.

## Variance

- Standard deviation increases with sample size but remains proportionally small.
- Larger samples produce smoother, more stable key ratios.

## Conclusion

This simulation successfully demonstrates: Implementation of a BB84-style QKD protocol

- Extraction of sifted keys
- Statistical behavior consistent with theoretical expectations

The visualizations confirm a stable and predictable relationship between sample size and final key length.