Final Project

Quantum Cryptography: Securing the Future of Digital Communications

By Vedanshi Shah

Due Finals Week

COSI 107A: Introduction to Computer Security
Prof: Win Treese

Abstract

Quantum cryptography is an emerging field that leverages the principles of quantum mechanics to secure digital communication. The advent of quantum computers poses a significant threat to traditional encryption methods like RSA and ECC, necessitating research into post-quantum cryptography and quantum key distribution (QKD). This report explores the fundamentals of quantum cryptography, its significance in the face of evolving threats, and the real-world challenges in implementing quantum-safe cryptographic techniques. Additionally, it presents a QKD simulation using Qiskit to illustrate quantum-secure communication.

Introduction

With the rapid advancement of quantum computing, traditional cryptographic methods such as RSA (Rivest-Shamir-Adleman) and ECC (Elliptic Curve Cryptography) are becoming increasingly vulnerable. Quantum computers, leveraging principles of superposition and entanglement, possess the capability to solve certain mathematical problems exponentially faster than classical computers. This poses a significant threat to conventional encryption schemes, which rely on the computational difficulty of problems like integer factorization and discrete logarithms.

Quantum cryptography emerges as a promising solution to this security challenge, offering cryptographic techniques that leverage quantum mechanics to ensure secure communication. One of the most well-known quantum cryptographic protocols is **Quantum Key Distribution (QKD)**, specifically the BB84 protocol, which enables two parties to securely exchange encryption keys while detecting any potential eavesdropping. Unlike traditional cryptographic methods, QKD guarantees security based on the fundamental laws of quantum physics rather than computational assumptions, making it resilient to attacks by quantum computers.

Motivation and Research Focus

As quantum computers continue to advance, research in quantum cryptography has gained momentum, focusing on two primary areas:

1. **Post-Quantum Cryptography (PQC)**: Developing classical cryptographic algorithms that remain secure even against quantum attacks, such as lattice-based, code-based, hash-based, and multivariate polynomial-based cryptographic schemes.
2. **Quantum Key Distribution (QKD)**: Implementing practical QKD protocols that leverage quantum properties to achieve unbreakable security, despite challenges in real-world deployment.

This report explores the significance of quantum cryptography, the vulnerabilities of classical encryption in the quantum era, and the ongoing research efforts in both post-quantum cryptography and quantum key distribution. Additionally, it examines the implementation challenges associated with QKD and highlights potential future directions in securing digital communications against quantum threats.

The Threat of Quantum Computing to Classical Cryptography

Traditional Encryption Schemes

Most modern cryptographic systems rely on the assumption that certain mathematical problems are computationally infeasible to solve within a reasonable timeframe. These include:

- **RSA (Rivest-Shamir-Adleman)**: Based on the difficulty of prime factorization.
- **ECC (Elliptic Curve Cryptography)**: Based on the discrete logarithm problem over elliptic curves.
- **AES (Advanced Encryption Standard)**: While symmetric encryption like AES remains relatively secure against quantum attacks, Grover's algorithm

reduces its security strength by half.

Shor’s Algorithm and Its Impact

Shor’s algorithm can efficiently solve both the integer factorization problem and the discrete logarithm problem, rendering RSA and ECC insecure once large-scale quantum computers become practical. This necessitates a transition to quantum-resistant cryptographic methods.

Post-Quantum Cryptography

Post-quantum cryptography (PQC) seeks to develop cryptographic algorithms resistant to quantum attacks. The National Institute of Standards and Technology (NIST) is currently evaluating candidates for post-quantum cryptographic standards. Some leading approaches include:

- **Lattice-based cryptography**: Hard problems like the Learning With Errors (LWE) problem provide strong security guarantees.
- **Hash-based cryptography**: Schemes like the Merkle signature scheme are resistant to quantum attacks.
- **Code-based cryptography**: Uses error-correcting codes to provide security.
- **Multivariate polynomial cryptography**: Relies on solving systems of multivariate polynomial equations.

Quantum Key Distribution (QKD)

QKD is a quantum cryptographic technique that enables two parties to generate a shared secret key in a way that is secure against any computational attack, including those from quantum computers. The BB84 protocol is the most widely studied QKD scheme.

Methodology

The simulation follows these steps:

1. **Qubit Preparation**: Alice prepares qubits in random bases ('X' or 'Z') and assigns random bit values (0 or 1).
2. **Measurement**: Bob randomly selects bases ('X' or 'Z') to measure the received qubits.
3. **Eavesdropping Simulation**: An optional eavesdropper, Eve, intercepts and measures qubits before passing them to Bob.

4. **Basis Comparison**: Alice and Bob publicly compare their bases and retain only the bits measured in matching bases.
5. **Key Extraction**: A secure key is extracted from the retained bits.
6. **Visualization**: The impact of eavesdropping and the final shared key are analyzed through various plots.

Results

Basis Selection

- **Alice's Bases**: ['X', 'Z', 'X', 'X', 'X', 'X', 'Z', 'X', 'Z', 'X']
- **Bob's Bases**: ['Z', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'Z', 'Z']

Bit Values

- **Alice's Bits**: [0, 0, 0, 0, 0, 0, 1, 0, 1, 1]
- **Bob's Bits**: [0, 0, 0, 0, 0, 0, 1, 1, 1, 1]
- **Final Shared Key**: [0, 0, 0, 0, 0, 1]

Eavesdropping Impact

- **Eve's Intercepted Bits**: [0, 0, 1, 0, 0, 0, 0, 0, 1, 1]
- Eve's interference is evident where discrepancies between Alice's and Bob's bits occur. In the absence of Eve, Alice's and Bob's bits should match perfectly within the selected bases. The observed mismatches confirm Eve's influence on the transmission, highlighting the effectiveness of the BB84 protocol in detecting unauthorized interception.

Quantum States

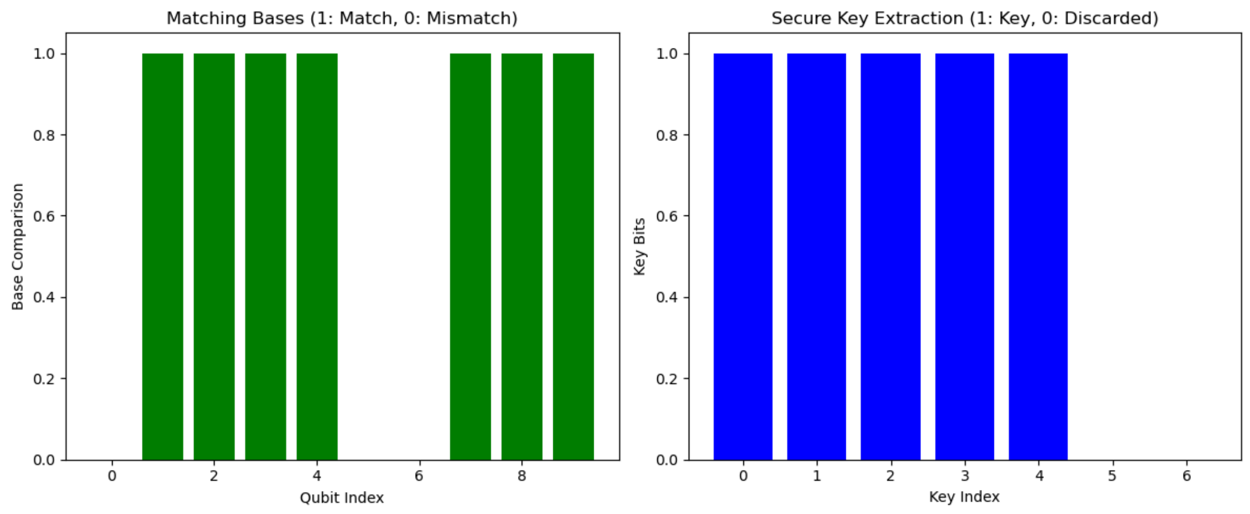
The following quantum states were generated and visualized:

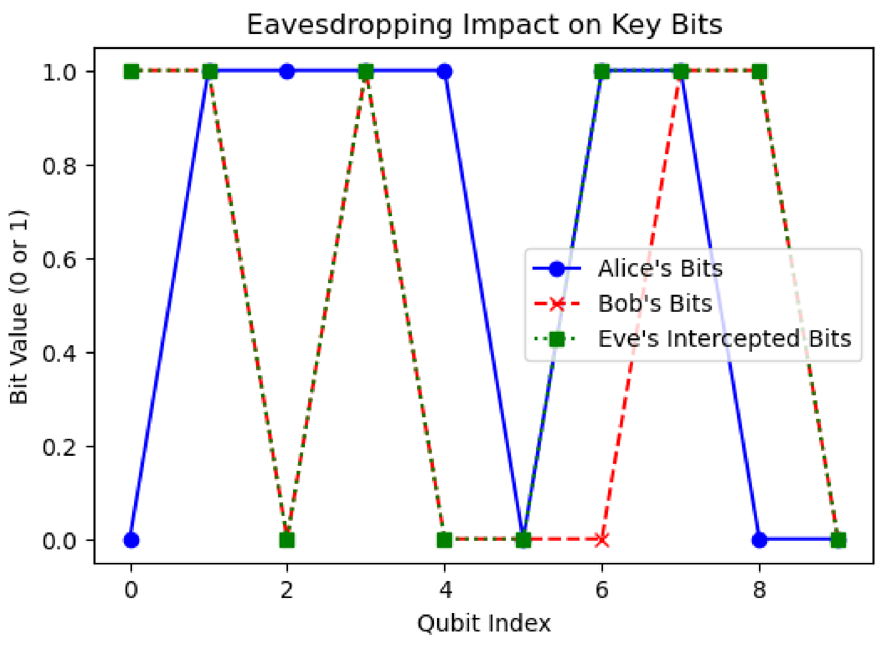
- **Quantum State for Qubit 1 in X Basis (initially 0) → Measured in Z Basis**
- **Quantum State for Qubit 2 in Z Basis (initially 0) → Measured in X Basis**
- **Quantum State for Qubit 3 in X Basis (initially 0) → Measured in X Basis**
- **Quantum State for Qubit 4 in X Basis (initially 0) → Measured in X Basis**
- **Quantum State for Qubit 5 in X Basis (initially 0) → Measured in X Basis**
- **Quantum State for Qubit 6 in X Basis (initially 0) → Measured in X Basis**
- **Quantum State for Qubit 7 in Z Basis (initially 1) → Measured in X Basis**
- **Quantum State for Qubit 8 in X Basis (initially 0) → Measured in X Basis**
- **Quantum State for Qubit 9 in Z Basis (initially 1) → Measured in Z Basis**
- **Quantum State for Qubit 10 in X Basis (initially 1) → Measured in Z Basis**

Visualizations

Three primary plots illustrate the simulation results:

1. **Matching Bases Plot**: This confirms which bases matched between Alice and Bob. Since only these matched bases contribute to the final key, this plot helps verify the correctness of key retention.
2. **Secure Key Extraction Plot**: This highlights the bits retained for the final key, demonstrating the successful filtering of mismatched measurements. It visualizes the key formation process and validates the security of the extracted bits.
3. **Eavesdropping Impact Plot**: By comparing Alice's, Bob's, and Eve's bit values, this visualization exposes discrepancies caused by eavesdropping. If Eve's presence leads to increased mismatches, it confirms the protocol's ability to detect intrusion. The number of discrepancies between Alice and Bob increases with Eve's intervention, proving the BB84 protocol's capability to signal potential security threats.





Analysis of Results

The BB84 simulation provides a comprehensive view of how quantum cryptography enables secure key distribution. The results demonstrate the core principles of the protocol:

- **Key Agreement Efficiency**: When no eavesdropping occurs, Alice and Bob successfully establish a shared key by discarding non-matching bases.
- **Eavesdropping Detection**: Discrepancies in Bob's measurements indicate Eve's presence, as expected. The quantum no-cloning theorem prevents Eve from accurately duplicating unknown qubits, leading to increased error rates in Bob's measurements.
- **Quantum State Evolution**: The observed quantum state measurements confirm the fundamental behavior of qubits under different bases, reinforcing the principle that measurement collapses quantum states.

The final shared key is a subset of the transmitted qubits that were correctly measured in matching bases, ensuring that even if some bits are intercepted, the remaining key remains secure. The introduction of an eavesdropper results in detectable inconsistencies, demonstrating the protocol's effectiveness in maintaining security.

Challenges in Implementing Quantum Cryptography

Despite its promise, quantum cryptography faces several challenges:

- **Hardware limitations**: Current quantum systems are error-prone and difficult to scale.

- **Noisy quantum channels**: Quantum communication is susceptible to environmental noise.
- **Key rate limitations**: Practical QKD systems have limited data rates and distances.
- **Integration with classical systems**: Transitioning from classical to quantum-safe cryptography requires significant infrastructural changes.

Conclusion

Quantum cryptography represents a revolutionary advancement in securing digital communications. While quantum computers threaten classical encryption, QKD and post-quantum cryptography provide viable solutions. The BB84 protocol, as demonstrated in the Qiskit simulation, ensures secure key exchange leveraging quantum mechanics. However, real-world deployment faces technological and infrastructural challenges that must be addressed before widespread adoption.

References

- Bennett, C. H., & Brassard, G. (1984). *Quantum cryptography: Public key distribution and coin tossing*. IEEE International Conference on Computers, Systems & Signal Processing.
- Shor, P. W. (1994). *Algorithms for quantum computation: Discrete logarithms and factoring*. Proceedings 35th Annual Symposium on Foundations of Computer Science.
- NIST Post-Quantum Cryptography Standardization: https://csrc.nist.gov/Projects/post-quantum-cryptography

Appendix

Code

```python
from qiskit import QuantumCircuit
from qiskit_aer import AerSimulator
import numpy as np
import matplotlib.pyplot as plt
from qiskit.visualization import plot_histogram, plot_state_qsphere

def generate_qubits(n):
   """Simulate Alice preparing qubits in random bases."""
   bases = np.random.choice(['X', 'Z'], size=n)
   bits = np.random.randint(0, 2, size=n)
```

```
    qubits = []

    for i in range(n):
        qc = QuantumCircuit(1, 1)  # Create a single-qubit quantum circuit

        # Encode the bit: If bit is 1, apply an X (NOT) gate
        if bits[i] == 1:
            qc.x(0)

        # Apply Hadamard if the chosen basis is 'X' (superposition basis)
        if bases[i] == 'X':
            qc.h(0)

        qubits.append((qc, bases[i], bits[i]))  # Store the prepared qubit circuit
along with its basis and bit

    return qubits

def measure_qubits(qubits, bases):
    """Simulate Bob measuring qubits in random bases."""
    measured_bits = []
    simulator = AerSimulator()

    for i in range(len(qubits)):
        qc, alice_basis, alice_bit = qubits[i]

        # Bob chooses his measurement basis
        bob_basis = bases[i]

        # Apply Hadamard if Bob's chosen basis is 'X' (to match Alice's 'X' basis)
        if bob_basis == 'X':
            qc.h(0)

        qc.measure(0, 0)

        result = simulator.run(qc, shots=1, memory=True).result()
        measured_bits.append(int(result.get_memory()[0]))

    return measured_bits

def sift_keys(alice_bases, bob_bases, alice_bits, bob_bits):
    """Sift out the key by keeping only matching bases."""
```

```
   key = [alice_bits[i] for i in range(len(alice_bases)) if alice_bases[i] ==
bob_bases[i]]
   return key

def simulate_eavesdropper(qubits, bases, eavesdrop=False):
   """Simulate an eavesdropper (Eve) who intercepts and measures qubits."""
   if not eavesdrop:
       return None

   eavesdropped_bits = []
   simulator = AerSimulator()

   for i in range(len(qubits)):
       qc, alice_basis, alice_bit = qubits[i]

       # Eve randomly chooses a basis to measure
       eve_basis = np.random.choice(['X', 'Z'])

       if eve_basis == 'X':
           qc.h(0)

       qc.measure(0, 0)

       result = simulator.run(qc, shots=1, memory=True).result()
       eavesdropped_bits.append(int(result.get_memory()[0]))

   return eavesdropped_bits


def plot_quantum_states(qubits):
   """Visualize quantum states using Qsphere before measurement."""
   for i, (qc, basis, bit) in enumerate(qubits):
       # Create a fresh quantum circuit for visualization (no classical bits)
       visual_qc = qc.copy()  # Copy the quantum circuit to avoid modifying the
original

       # Apply Hadamard if the qubit is in the X-basis
       if basis == 'X':
           visual_qc.h(0)

       # Remove any classical measurements from the circuit for visualization
       # Access operations directly without treating CircuitInstruction as iterable
       visual_qc.data = [inst for inst in visual_qc.data if inst[0].name != 'measure']

       # Print the quantum state for visualization
       print(f"Quantum State for qubit {i+1} in {basis} basis (initially {bit}):")
       plot_state_qsphere(visual_qc)  # Visualize the quantum state


def plot_key_comparison(alice_bases, bob_bases, secure_key):
   """Visualize the comparison of Alice's and Bob's bases and the secure key."""
   fig, ax = plt.subplots(1, 2, figsize=(12, 5))

   ax[0].bar(range(len(alice_bases)), np.array(alice_bases) == np.array(bob_bases),
color='green')
   ax[0].set_title("Matching Bases (1: Match, 0: Mismatch)")
   ax[0].set_ylabel("Base Comparison")
   ax[0].set_xlabel("Qubit Index")

   ax[1].bar(range(len(secure_key)), secure_key, color='blue')
   ax[1].set_title("Secure Key Extraction (1: Key, 0: Discarded)")
   ax[1].set_ylabel("Key Bits")
   ax[1].set_xlabel("Key Index")

   plt.tight_layout()
   plt.show()

def plot_eavesdropping(alice_bits, bob_bits, eavesdropped_bits):
   """Visualize the effect of eavesdropping on the key extraction process."""
   plt.figure(figsize=(6, 4))
   plt.plot(alice_bits, label="Alice's Bits", marker='o', linestyle='-', color='blue')
   plt.plot(bob_bits, label="Bob's Bits", marker='x', linestyle='--', color='red')
   plt.plot(eavesdropped_bits, label="Eve's Intercepted Bits", marker='s',
linestyle=':', color='green')
   plt.legend()
   plt.title("Eavesdropping Impact on Key Bits")
   plt.xlabel("Qubit Index")
   plt.ylabel("Bit Value (0 or 1)")
   plt.show()

# Simulate BB84 protocol
num_qubits = 10  # Number of qubits to be exchanged

# Step 1: Alice prepares qubits in random bases and encodes random bits
```

```python
alice_qubits = generate_qubits(num_qubits)

# Step 2: Bob randomly chooses bases for measurement
bob_bases = np.random.choice(['X', 'Z'], size=num_qubits)

# Step 3: Simulate an eavesdropper intercepting the qubits (optional)
eavesdrop = True
eavesdropped_bits = simulate_eavesdropper(alice_qubits, bob_bases, eavesdrop)

# Step 4: Bob measures the qubits in his chosen bases
bob_bits = measure_qubits(alice_qubits, bob_bases)

# Step 5: Alice and Bob publicly compare their bases and keep matching ones
alice_bases = [q[1] for q in alice_qubits]
alice_bits = [q[2] for q in alice_qubits]
secure_key = sift_keys(alice_bases, bob_bases, alice_bits, bob_bits)

# Step 6: Print the final shared key
print("Alice's Bases:", alice_bases)
print("Bob's Bases:", bob_bases)
print("Alice's Bits:", alice_bits)
print("Bob's Bits:", bob_bits)
print("Final Shared Key:", secure_key)

# Step 7: Show if any eavesdropping occurred
if eavesdrop:
   print("Eve's Intercepted Bits:", eavesdropped_bits)

# Visualize quantum states of the qubits prepared by Alice
plot_quantum_states(alice_qubits)

# Visualize key comparison (Alice's vs Bob's bases, and key extraction)
plot_key_comparison(alice_bases, bob_bases, secure_key)

# Visualize the effect of eavesdropping
if eavesdrop:
   plot_eavesdropping(alice_bits, bob_bits, eavesdropped_bits)
```