Computing_Project

November 25, 2024

1 Introduction to Quantum Cryptography

Nihal Faiz

K21004658

K21004658@kcl.ac.uk

Modules and Packages used:

[ ]:
```
import numpy as np
from qiskit_aer import AerSimulator
from qiskit import QuantumCircuit, transpile
from qiskit_aer.noise import NoiseModel, depolarizing_error
```

1.1 Introduction to Quantum Computing

Before looking into specific examples and implications of Quantum Cryptography, it is worth giving a general overview of quantum computing. Quantum computing leverages quantum mechanics to tackle problems beyond the reach of classical computers. Richard Feynman first proposed the concept in 1982 [*1*], envisioning the use of quantum systems for computational purposes. While classical and quantum computers share certain foundational elements, such as chips, logic gates, and binary representation, their fundamental difference lies in how they encode and process information. Classical bits represent either a 0 or 1, whereas quantum bits, or qubits, can represent both 0 and 1 simultaneously due to superposition.

Superposition allows qubits to exist in a state that is a combination of both $|0\rangle$ and $|1\rangle$, providing quantum computers with vastly greater capacity to store and manipulate information. When measured, a qubit collapses to either 0 or 1, but until that moment, it effectively holds both possibilities, allowing quantum computers to perform multiple calculations at once. This property is key to their potential for solving certain problems exponentially faster than classical computers.

Another fundamental principle in quantum computing is entanglement, a phenomenon where two or more qubits become interconnected such that the state of one directly determines the state of the other, regardless of distance.

Quantum gates are used to manipulate qubits, like logic gates in classical computing. These gates, such as the Hadamard gate (which induces superposition) and the CNOT gate (which creates entanglement), exploit quantum properties to perform operations on qubits.

2 What is quantum cryptography?

Quantum cryptograhpy leverages quantum mechanics to create a communication system that is secure enough to withstand attacks from classical and quantum computers. The primary focus of quantum cryptography is quantum key distribution (QKD), which enables to locations to a secret key that can be used to encyrpt or decrpyt messages. Additionally, quantum computing poses a significant threat to traditional encryption methods, such as Rivest–Shamir–Adleman (RSA), as algorithms like Shor's Algorithm could efficiently break their security — a concept explored later in this project.

2.1 Quantum Key Distribution

QKD is vital for quantum cryptography. **BB84** is the most widely used protocol for QKD, it was initially proposed in 1984 by Charles Bennett and Gilles Brassard [*2*]. QKD relies on the following fundamental principles of quantum mechanics.

- **Heisenberg Uncertainty Principle** In quantum systems, only one property of a pair of conjugate properties can be known for certain. This principle gets taken advantage of in quantum cryptography by using the polarisation of photons on different bases as the conjuagte properties.
- **Superposition** : As previously mentioned, quantum particles can exist in multiple states simultaneously until the particle has been measured. This property enables the ability to encode information in multiple bases, such as different polarisation directions i.e. horizontal/ vertical or diagonal polarization of photons.
- **No-cloning** : This theorem states that it is impossible to create an independent and identical copy of an unkown quantum system. This property forbids eavesdroppers from creating copies of the transmitted cryptographic key. More mathematically put, the theorem says that there is no unitary operation that can take an unknown arbitary quantum state $|\psi\rangle$ and produce a copy of it, i.e, it is impossible to construct a unitary operator that acts as:

$$U(|\psi\rangle \otimes |0\rangle) = |\psi\rangle \otimes |\psi\rangle \tag{1}$$

- **Measurement disturbs state** : If the an eavesdropper tries to measure the quantum states that are being exchanged, they induce detectable changes due to the disturbance caused by the measurement.
- **Entanglement** : Two quantum particles can be entangled, regardless of the distance between them. When a property is measured in a particle, a correlated state of the property will appear on the other.

2.2 BB84 Protocol

The BB84 protocol consists of a sender (Alice) and a receiver (Bob), they share a sequence of photons which are encoded with random polarization states. The no-cloning theorem eneures that an eavesdropper (Eve) cannot measure the sequence without disturbing the photons, making Eve detectable. This is only true if there is no error in the system, otherwise Eve will be undetectable. The protocol can be divided into 3 primary components, communication over a quantum channel, classcal communication, and eavesdropping detection.

2.2.1 Quantum Communication

In the initial phase, Alice and Bob communicate using a quantum channel to establish a secret key. Alice would generate two random strings, sequence of bits and measurement bases. The bases are chosen at random, either a rectilinear basis (horizontal or vertical polarisation denoted as +) or a diagonal basis (45° or 135° polarisation, denoted as an x)

These bases are used by Alice to encode each bit onto a photon. For example, if she tries to transmit a bit of "0" and choose the rectilinear basis, she may encode the bit as a horizonatally polarized photon. These photons are then sent sequentially through the quantum channel (optical fibre), to Bob.

When Bob recieves these photons, he chooses a random measurement basis for each one. If the bases match, he correctly measures the photon and is able to identify the transmitted bit. However, if he uses a different basis, he will measure incorrectly and, as a result, would obtain a random bit.

[ ]:
```
n = 64 # define the number of qubits

#  Alice's random bits and bases
alice_bits = np.random.randint(2, size=n)
alice_bases = np.random.randint(2, size=n)

#  Bob's random bases
bob_bases = np.random.randint(2, size=n)
```

`alice_bases` is a random array representing Alice's choice of measurement bases:

- 0: The + basis (computational, i.e. $|0\rangle$ and $|1\rangle$)
- 1: The x basis (Hadamard basis, i.e. $|+\rangle$ and $|-\rangle$)

[ ]:
```
print("Alice's bases:   ", alice_bases)
print("Alice's bits:    ", alice_bits)
print("Bob's bases:     ", bob_bases)
```

```
Alice's bases:    [1 0 1 0 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 0 1 0 1 1
0 0 0 1 1 1
 1 0 1 1 0 0 1 1 1 1 0 1 1 0 1 0 0 1 1 1 1 0 1 0 1 1 1]
Alice's bits:     [0 0 1 0 1 0 0 1 1 1 1 1 0 1 0 0 0 0 1 1 0 1 1 0 1 0 1 1 0 0 0
0 0 0 1 0 0
 1 0 1 0 0 0 0 0 1 1 0 1 0 1 1 0 0 0 0 0 0 0 1 0 0 0 1]
Bob's bases:      [1 0 0 1 0 1 0 0 1 1 0 0 0 0 1 0 1 0 0 1 1 1 1 1 1 1 0 0 1 0 1
0 0 0 1 0 0
 0 0 1 0 0 0 1 0 0 0 1 1 0 1 1 0 1 1 1 0 1 0 1 1 0 1 0]
```

[ ]:
```
received_bits = [] # to store the received bits
backend = AerSimulator() # simulates the quantum operations

for i in range(n):
    qc = QuantumCircuit(1, 1)
    # Encode Alice's bit into her chosen basis
```

```
    if alice_bases[i] == 0:  # + basis
        if alice_bits[i] == 1:
            qc.x(0)  # |1
    else:  # × basis
        if alice_bits[i] == 0:
            qc.h(0)  # |+
        else:
            qc.x(0)
            qc.h(0)  # |-

    # Bob's measurement
    if bob_bases[i] == 1:  # Measure in × basis
        qc.h(0)
    qc.measure(0, 0)

    # Run the circuit directly on the backend
    job = backend.run(qc, shots=10)
    result = job.result()
    counts = result.get_counts()
    measured_bit = int(list(counts.keys())[0])
    received_bits.append(measured_bit)
```

- When Alice uses the + basis (`alice_bases[i] == 0`):
  - if the bit she wants to encode is 0, the qubit stays in its default state $|0\rangle$
  - if the bit she wants to encode it 1, she flips the bit using an X gate, which would result in the $|1\rangle$
- When Alice uses the x basis (`alice_bases[i] == 1`):
  - if the bit she wants to encode is 0, she applies the Hadammard gate, transforming it in to the $|+\rangle$
  - if the bit is 1, she first flips the qubit with an X gate and then applies a Hadammard, producing a $|-\rangle$

2.2.2 Classical Communication

After this quantum communication, the sender and reciever engage in classical communication where the reciever shares which of the bases they used for each of the photons without sharing the measured value of the bits. The sender then confirms which of the bases matched. Using this information, they agree to keep only the bits for which receivers matched the senders, disregarding the rest. This process produces a shorter, identical bit-string shared between the sender and the receiver, which is known as the sifted key.

[ ]:
```
sifted_key = []
for i in range(n):
    if alice_bases[i] == bob_bases[i]:  # Keep bits where bases match
        sifted_key.append(received_bits[i])

# Display results
print("Alice's bits:    ", alice_bits)
```

```
print("Bob's received bits:", received_bits)
print("Sifted key:      ", sifted_key)
```

```
Alice's bits:     [0 0 1 0 1 0 0 1 1 1 1 1 0 1 0 0 0 0 1 1 0 1 1 0 1 0 1 1 0 0 0
0 0 0 1 0 0
 1 0 1 0 0 0 0 0 1 1 0 1 0 1 1 0 0 0 0 0 0 0 1 0 0 0 1]
Bob's received bits: [0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0,
0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1,
0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1]
Sifted key:       [0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0,
1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0]
```

2.2.3 Detecting Eavesdropping

The security of the BB84 protocol relies on the fundamental principle that measuring a quantum system inevitably disturbs it. To verify the security of their communication, Alice and Bob reveal and compare a subset of bits from their sifted key over a classical channel. If the revealed bits match, they can confidently conclude that the transmission was secure. However, if the revealed bits do not match, it suggests the presence of an eavesdropper, commonly referred to as Eve.

Eve's detection is possible because intercepting and measuring a photon requires her to guess the basis used by Alice. If she guesses incorrectly, her measurement collapses the quantum state, introducing errors that are detectable by Alice and Bob when they compare their revealed bits.

In the simulation below, we also introduce noise, which simulates the effects of imperfections in the quantum communication channel or hardware. Noise can arise from various sources, such as environmental interference, thermal fluctuations, or imperfections in quantum devices. It manifests as random errors in the transmitted qubits, which could mimic the effects of eavesdropping.

[ ]:
```
# Eve's random bases and measurements
eve_bases = np.random.randint(2, size=n)  # Eve's random bases
eve_measured_bits = []

bob_measured_bits = []  # Collect all of Bob's measured bits

# Simulator backend
backend = AerSimulator()

# Add depolarizing noise to the simulation
noise_model = NoiseModel()
depolarizing_noise = depolarizing_error(0.1, 1)  # 10% depolarizing noise for␣
 ↪single qubits
noise_model.add_all_qubit_quantum_error(depolarizing_noise, ['h', 'x',␣
 ↪'measure'])

for i in range(n):
    qc = QuantumCircuit(1, 1)
    # Alice prepares the qubit
    if alice_bases[i] == 0:  # + basis
```

```
        if alice_bits[i] == 1:
            qc.x(0)  # |1
    else:  # × basis
        if alice_bits[i] == 0:
            qc.h(0)  # |+
        else:
            qc.x(0)
            qc.h(0)  # |-

    # Eve measures in her random basis
    eve_qc = qc.copy()
    if eve_bases[i] == 1:  # × basis
        eve_qc.h(0)
    eve_qc.measure(0, 0)

    eve_result = backend.run(transpile(eve_qc, backend),␣
↪noise_model=noise_model, shots=100).result()
    eve_counts = eve_result.get_counts()
    eve_measured_bit = int(max(eve_counts, key=eve_counts.get))
    eve_measured_bits.append(eve_measured_bit)

    # Eve resends the qubit based on her measurement
    qc = QuantumCircuit(1, 1)
    if eve_bases[i] == 0:  # Resend in + basis
        if eve_measured_bit == 1:
            qc.x(0)
    else:  # Resend in × basis
        if eve_measured_bit == 0:
            qc.h(0)
        else:
            qc.x(0)
            qc.h(0)

    # Bob measures the received qubit
    if bob_bases[i] == 1:  # Measure in × basis
        qc.h(0)
    qc.measure(0, 0)



    bob_result = backend.run(transpile(qc, backend), noise_model=noise_model,␣
↪shots=100).result()
    bob_counts = bob_result.get_counts()
    bob_measured_bit = int(max(bob_counts, key=bob_counts.get))
    bob_measured_bits.append(bob_measured_bit)  # Store Bob's measured bit
```

Eve uses her own randomly chosen measurement bases to measure the qubits. Since Eve doesn't

know which bases Alice used, she has a 50% chance of selecting the correct one for each qubit. After measuring the qubits, Eve then sends them onward to Bob.

This interception by Eve changes the original quantum state of the qubits. Because quantum measurements are inherently disruptive, once Eve measures a qubit, it collapses to a definite state corresponding to her measurement. When Eve forwards the qubits to Bob, Bob receives qubits that may have already been altered by Eve's incorrect measurement.

Consequently, when Bob measures the qubits, he is now working with a set of quantum states that might differ from the original states that Alice sent. Bob, unaware of Eve's interference, measures the qubits using his own randomly chosen bases. This results in a higher error rate in the comparison process between Alice and Bob since Eve's presence disrupts the consistency of the key. When Alice and Bob compare a subset of their bits publicly to check for discrepancies, the higher-than-expected error rate will indicate the presence of an eavesdropper like Eve.

[ ]:
```
shared_key = []
revealed_indices = np.random.choice(range(n), size=n // 2, replace=False)  #␣
 ↪Publicly reveal half the bits
errors_detected = 0

for i in range(n):
    if alice_bases[i] == bob_bases[i]:  # Keep bits where bases match
        if i in revealed_indices:
            # Publicly compare the revealed bits
            if alice_bits[i] != bob_measured_bits[i]:
                errors_detected += 1
        else:
            # Keep bits not revealed
            shared_key.append(bob_measured_bits[i])

# Calculate the error rate
error_rate = errors_detected / len(revealed_indices)


if error_rate > 0.1:  # Threshold for detection (10% error rate)
    print("Eavesdropping detected! Key aborted.")
else:
    print("No eavesdropping detected. Key accepted.")
    print("Shared key: ", shared_key)

print(f"Error rate: {error_rate:.2%}")
```

```
Eavesdropping detected! Key aborted.
Error rate: 12.50%
```

The code cell above evaluates the process of establishing a shared key between Alice and Bob by checking when Bob correctly guesses the measurement basis compared to Alice's basis. It identifies bits where their bases match and separates them into revealed and unrevealed sets. For the revealed bits, Alice and Bob publicly compare their measured values. If the values differ, this indicates an

error, which could be caused by eavesdropping (by Eve) or noise. The error rate is calculated as the ratio of detected errors to the number of revealed bits. If this error rate exceeds a predefined threshold of 10%, it suggests potential interference, and the key generation process is aborted. If the error rate is below the threshold, the shared key is accepted, consisting of the unrevealed bits measured by Bob. The process ensures that the shared key remains secure and reliable under normal conditions.

3 Schor's algorithm vs RSA Encryption

The greatest threat to cyber security is the possibility of breaking RSA encyrption with a powerul enough quantum computer. RSA encryption is widely implemented and is used to begin over 90% [4] of internet connections. The algorithm functions based on prime factorisation. It is an asymmetric encryption algorithm meaning a public and private key are used in the encryption and decryption process. The message in encyrpted by the sender using a public key and can only be decrypted by the reciever's private key. The sender encrypts the message using the product of two prime numbers. The product of the two numbers is the public key. The reciever has the two prime numbers that were multiplied togther - this is the private key. RSA pubic keys can be up to 617 [4] digits long which makes it computationally impossible for classical computers to prime factorise such a large number.

Peter Shor developed a quantum factoring algorithm that has the potential to break RSA encryption given a powerful enough quantum computer. The algorithm begins by selecting a random integer smaller than the number that you are looking to factorise, N. The classical computation of the greatest common divisor (GCD) between this random number and N checks if the number has already been factored accidentally. The quantum computers main role is to determine the period of a specific function related to the number that needs to be factorised. Based on this result, the algorithm finds the factors or decides whether a new random number needs to be chosen to be tested.

It is composed of three major components:

- Initial classical computation
- Quantum computation to find the period
- Final classical computation to derive the factors

The quantum components, specifically quantum phase estimation (QPE) and inverse quantum fourier transforms (iQFT) underpin the entire process. QPE is essential for determinig the period of the arithmetic function $ f(x) = a^x mod N $. After the QPE calculates the arithemtic function, the iQFT converts the quantum state into classical information that can be retrieved from the quantum circuit by measurement. These quantum processes enable the algorithm to theoretically calculate the factors of very large numbers.

3.1 Limitations

Although Shor's algorithm is extremely powerful, RSA encryption has not yet been broken mainly due to the physical restrictions on quantum computers. The largest number to be factorised using Shor's algorithim is 21 using only a 5 qubit quantum computer[5]. Companies such as D-wave have quantum computers with over 5000 qubtis. Despite this fact, reason for the lack of progression in factorising larger is not due to the quantity of qubits in a quantum computer, but actually a matter

of the number of stable qubits. Quantum computers are prone to error and noise which makes it diffiult to extract meaningful values.

4 Conclusion

This project explored quantum cryptography's potential, particularly the Quantum Key Distribution (QKD) through the BB84 protocol, and the implications of quantum computing with Shor's algorithm. While both represent groundbreaking advancements, they come with notable limitations.

The BB84 protocol offers robust security by leveraging quantum mechanical properties like superposition, entanglement, and the no-cloning theorem. However, its practicality faces significant challenges. Real-world implementation of this protocol is set back by a number of factors such as noise in quantum channels, environmental interference, and technological imperfections in quantum devices. These factors can introduce errors that can mask the intrusion of an eavesdropper. Additionally, the protocol relies on secure classical communication, which may itself become vulnerable without secure encryption.

Similarly, while Shor's algorithm poses a threat to classical encryption, its practical application is constrained by the current state of quantum hardware. Quantum computers today lack the stability and scale required to factorise large numbers efficiently. Issues such as qubit decoherence, error rates, and limited quantum coherence time hinder the algorithm's effectiveness. Currently Shor's algorithm has only been able to factorise numbers up to 21, far below the scale needed to compromise RSA encryption in real-world scenarios.

5 Bibliography

[1] Resonance (2024) The history of quantum computing you need to know [2024], The Quantum Insider. Available at: https://thequantuminsider.com/2020/05/26/history-of-quantum-computing/

[2] Quantum blockchains. Available at: https://www.quantumblockchains.io/qkd-protocol-simulation-with-qiskit/

[3] MR.Asif (2022) Quantum key distribution and BB84 protocol, Medium. Available at: https://medium.com/quantum-untangled/quantum-key-distribution-and-bb84-protocol-6f03cc6263c5

[4] Bose, A. (2024) Shor's algorithm and RSA encryption , Quantum Algorithms Institute. Available at: https://www.qai.ca/resource-library/shors-algorithm-and-rsa-encryptionnbsp#:~:text=Although%20Shor's%20algorithm%20is%20incredibly,factors%20of%203%20and%205.

[5] Skosana, U., Tame, M. Demonstration of Shor's factoring algorithm for N = 21 on IBM quantum processors. Sci Rep 11, 16599 (2021). https://doi.org/10.1038/s41598-021-95973-w

[6] qbit12 (2021) Tutorials/intermediate/the BB84 quantum cryptography algorithm.ipynb at main · qmunitytech/tutorials, GitHub. Available at: https://github.com/qmunitytech/Tutorials/blob/main/intermediate/The%20BB84%20Quantum%20Cryptograph

[7] M. H. Saeed, H. Sattar, M. H. Durad and Z. Haider, "Implementation of QKD BB84 Protocol in Qiskit," 2022 19th International Bhurban Conference on Applied Sciences and Technology (IBCAST), Islamabad, Pakistan, 2022, pp. 689-695, doi: 10.1109/IBCAST54850.2022.9990073.

[8] Booth, G., 2024. Quantum Information and Computing Lecture Notes: 7CCP3000. [Lecture notes] King's College London, Department of Physics, 5 November.