Quantum Key Distribution

Capstone Project

Introduction

In developing the encryption and decryption functions for my project, I focused on creating robust and efficient algorithms while considering the specific requirements of the BB84 quantum key distribution protocol and the *Rail Fence Cipher*. The goal was to ensure that my functions could securely encrypt and decrypt messages while maintaining compatibility with quantum computing principles.

When building the encrypt and decrypt functions, several considerations were taken into account:

1. Algorithm Design: I carefully designed the Rail Fence cipher algorithm to ensure that it can effectively encrypt and decrypt messages using a specified key.
2. Key Management: The functions were designed to handle keys securely, ensuring that the encryption key used in encryption matches the decryption key used in decryption. This is crucial for maintaining data integrity and confidentiality.

Difficulty Encountered

During the development process, I encountered a challenge:

Algorithm Complexity: The Rail Fence cipher, although conceptually simple, required careful implementation due to its zigzag pattern and key-dependent behaviour. Ensuring correctness and efficiency in handling varying key sizes posed a challenge.

Code Snippets:

```
def encryptRailFence(text, key):
        rail = [['\n' for i in range(len(text))]
                                        for j in range(key)]
        dir_down = False
        row, col = 0, 0

        for i in range(len(text)):
                if (row == 0) or (row == key - 1):
                        dir_down = not dir_down

                rail[row][col] = text[i]
                col += 1

                if dir_down:
                        row += 1
                else:
                        row -= 1
        result = []
        for i in range(key):
                for j in range(len(text)):
                        if rail[i][j] != '\n':
                                result.append(rail[i][j])
        return("" . join(result))
```

```
def decryptRailFence(cipher, key):
        rail = [['\n' for i in range(len(cipher))]
                                        for j in range(key)]

        dir_down = None
        row, col = 0, 0

        for i in range(len(cipher)):
                if row == 0:
                        dir_down = True
                if row == key - 1:
                        dir_down = False

                rail[row][col] = '*'
                col += 1

                if dir_down:
                        row += 1
                else:
                        row -= 1

        index = 0
        for i in range(key):
                for j in range(len(cipher)):
                        if ((rail[i][j] == '*') and
                        (index < len(cipher))):
                                rail[i][j] =
cipher[index]
                                index += 1

        result = []
        row, col = 0, 0
        for i in range(len(cipher)):

                if row == 0:
                        dir_down = True
                if row == key-1:
                        dir_down = False

                if (rail[row][col] != '*'):
                        result.append(rail[row][col])
                        col += 1
```

```
            if dir_down:
                row += 1
            else:
                row -= 1
    return("".join(result))
```

1. encryptRailFence(text, key)

Arguments:
text: The plaintext message to be encrypted.
key: The key used for the Rail Fence cipher encryption.
Returns:
Encrypted cipher text using the Rail Fence cipher.
Approach:
This function implements the Rail Fence cipher encryption technique.
It creates a rail matrix based on the key and the length of the plaintext message.
The plaintext characters are then filled into the rail matrix in a zigzag pattern.
The function returns the encrypted cipher text obtained from the rail matrix.

2. decryptRailFence(cipher, key)

Arguments:
cipher: The cipher text message to be decrypted.
key: The key used for the Rail Fence cipher decryption.
Returns:
Decrypted plaintext using the Rail Fence cipher.
Approach:
This function implements the Rail Fence cipher decryption technique.
It initialises a rail matrix based on the key and the length of the cipher text message.
The positions for cipher text characters in the rail matrix are marked according to the decryption pattern.
The function reads the characters from the rail matrix to reconstruct the original plaintext message.

Analysis: Impact of Eavesdropping

An eavesdropper in a quantum communication channel can significantly impact the encryption and decryption processes. In the BB84 protocol, for instance, unauthorised interception of qubits by an eavesdropper can lead to key compromise. Since quantum mechanics dictates that measuring a qubit alters its state irreversibly, any unauthorised measurement by an eavesdropper would disrupt the key distribution process.

For the Rail Fence cipher, eavesdropping can compromise the security of the encrypted message if the key used for encryption is intercepted. If an attacker gains access to both the cipher text and the decryption key, they can potentially decrypt the message and obtain the original plaintext.

Conclusion: Importance of Quantum Encryption

The development of quantum encryption techniques, such as those implemented in our project, is vital for the future of secure communication.

1. Unbreakable Security: Quantum encryption leverages the laws of quantum mechanics to provide theoretically unbreakable security.

2. Quantum Key Distribution: Protocols like BB84 enable secure key distribution between parties, guaranteeing that any eavesdropping attempts can be detected.

3. Resilience to Quantum Computing: As quantum computing capabilities advance, traditional encryption methods may become vulnerable to quantum attacks

In conclusion, the development and adoption of quantum encryption technologies are crucial for establishing a secure and resilient communication infrastructure in the era of quantum computing. These advancements will play a pivotal role in safeguarding sensitive information and fostering trust in digital interactions.