Encryption using QKD

Version 1.0

Manish Roy

April 21, 2024

Submitted By: `Manish Roy`

Index

- Encryption using QKD
  - Introduction
  - Project Details
    - Repository
    - Tools and Technologies
    - Project structure
  - Files
    - requirements.txt
    - src/__init__.py
    - src/decryption.py
    - src/encryption.py
    - main.py

Encryption using QKD

Introduction

This repository has been created as a *Capstone Project* for **The Coding School - Qubit by Qubit**'s 2nd Semester of session 2023-2024. The project demonstrates the usage of **Quantum Key Distribution** for communication of mesages between two parties, attempting to prevent any interference from an eavesdropper.

Project Details

Repository

https://github.com/mv3n0m/QxQ-Capstone.git

Tools and Technologies

- Python >= v3.9
- Cirq == v1.3.0 => An open source framework for programming quantum computers provided by Google

Project structure

```
src/
|--- __init__.py
|--- decryption.py
|--- encryption.py
requirements.txt
main.py
```

Files

requirements.txt

Contains list of dependencies to be installed before running the project.

To install

```
pip install -r requirements.txt
```

src/__init__.py

Contains basic functions and declarations as follows:

```python
import cirq
from random import choices

encode_gates = { 0: cirq.I, 1: cirq.X }
basis_gates = { 'X': cirq.H, 'Y': cirq.Y, 'Z': cirq.I }

get_qubits = lambda n: cirq.NamedQubit.range(n, prefix='q')


def text_to_bits(text):
    bits = ''.join(format(ord(char), '08b') for char in text)
    return [int(bit) for bit in bits]

def bits_to_text(bits):
    text = ''.join(chr(int(''.join(map(str, bits[i:i+8])), 2)) for i in
         range(0, len(bits), 8))
    return text
```

src/decryption.py

Contains function for decryption

```python
import cirq
from . import basis_gates, get_qubits

def decrypt_message(encrypted_circuit, num_bits, private_key, bases):
    qubits = get_qubits(num_bits)

    circuit = cirq.Circuit()
    for idx in range(num_bits):
        basis_value = bases[idx]
        basis_gate = basis_gates[basis_value]
        qubit = qubits[idx]
        circuit.append(basis_gate(qubit))

    circuit.append(cirq.measure(qubits, key=private_key))
    bb84_circuit = encrypted_circuit + circuit
    sim = cirq.Simulator()
    results = sim.run(bb84_circuit)
    key = results.measurements[private_key][0]

    return key
```

src/encryption.py

Contains function for encryption

```python
import cirq
from . import encode_gates, basis_gates, get_qubits

def encrypt_message(message_bits, num_bits, bases):
    qubits = get_qubits(num_bits)

    circuit = cirq.Circuit()
    for idx in range(num_bits):
        encode_value = message_bits[idx]
        encode_gate = encode_gates[encode_value]
        basis_value = bases[idx]
        basis_gate = basis_gates[basis_value]
        qubit = qubits[idx]
        circuit.append(encode_gate(qubit))
        circuit.append(basis_gate(qubit))

    return circuit
```

main.py

Contains the driver code to run the project and manage inputs.

To run

```
python main.py
```

`main.py` imports the above modules to perform operations like converting message to bits, encryption, decryption, comparing bases and checking for eavesdropper.

- importing modules

```python
from src import text_to_bits, bits_to_text
from src.encryption import encrypt_message
from src.decryption import decrypt_message
from random import choices
```

- converting message to bits

```python
message = "My journey with The Coding School - Qubit by Qubit, has been
     truly unique and exceptional."

message_bits = text_to_bits(message)
num_bits = len(message_bits)
```

- getting encryption circuit

```python
base_options = ['Z', 'X']

encryption_bases = choices(base_options, k=num_bits)
encrypted_circuit = encrypt_message(message_bits, num_bits,
    encryption_bases)
```

- getting decrypted bits

```python
private_key = 'some random key'
decryption_bases =  choices(base_options, k=num_bits)
decrypted_bits = decrypt_message(encrypted_circuit, num_bits,
    private_key, decryption_bases)
```

- comparing bases

```python
private_key = 'some random key'
decryption_bases =  choices(base_options, k=num_bits)
decrypted_bits = decrypt_message(encrypted_circuit, num_bits,
    private_key, decryption_bases)
```

- checking for an eavesdropper

```python
if encrypted[0] == decrypted[0]:
    encrypted = encrypted[1:]
    decrypted = decrypted[1:]
    print('Keys are not compromised.')
else:
    print('Eve was listening, we need to use a different channel!')
```

If the first bits of the above encrypted and decrypted bits are equal, we can conclude that the communication went through without any interference of an eavesdropper. If the eavesdropper had tried to read the message, the bits would have been compromised alarming the two parties.