CS610 - Project 1: QKD-Encrypted LLM for Secure Local Use

Abraham J. Reines

August 25, 2024

Contents

1 Introduction

This project involves deploying a small pre-trained language model (GPT-Neo) on the 'stu.cs.jmu.edu' server and using Quantum Key Distribution (QKD) to securely encrypt communications between any client device and the server. The goal is to explore advanced cryptographic techniques and their application in secure communications, aligning with the course's focus on networking and security.

2 High-Level Design

2.1 System Architecture

The system is designed as a client-server architecture:

- **Server Side:** The server hosts the pre-trained LLM (GPT-Neo) and handles encrypted communication with clients. It runs on 'stu.cs.jmu.edu'.
- **Client Side:** Clients communicate with the server over a secure network. They use QKD to exchange encryption keys, which are then used to encrypt and decrypt communications.

2.2 Quantum Key Distribution (QKD) Simulation

A QKD simulation will be implemented to securely exchange encryption keys between the client and server. This simulation mimics the key exchange process in a quantum communication channel, ensuring the security of the encryption keys even against quantum attacks.

2.3 Encryption and Decryption

- The LLM model is stored in an encrypted format on the server.
- When a client connects, the QKD process is initiated to securely exchange the encryption key.
- The client uses the key to decrypt the LLM model for local use, and all subsequent communications are encrypted with this key.

3 Implementation Details

3.1 Programming Language

The project is implemented in Python, which is supported on 'stu.cs.jmu.edu'. Python is chosen due to its rich ecosystem of libraries for both machine learning and cryptography.

3.2 Specialized Libraries

- **Hugging Face Transformers:** Used to deploy the GPT-Neo model.
- **PyCryptodome:** Used for encryption and decryption processes.
- **Qiskit:** Used to simulate the QKD process.

3.3 Key Modules

- **server.py:** Manages the LLM and handles client connections.
- **client.py:** Initiates connections with the server, handles QKD, and decrypts the LLM model for use.
- **qkd.py:** Implements the QKD simulation, ensuring secure key exchange between client and server.

4 Compiling and Running the Project

4.1 Setup Instructions

- Ensure Python is installed on both the server and client machines.
- Install the required libraries using pip:

```
pip install transformers pycryptodome qiskit
```

- Place the 'server.py', 'client.py', and 'qkd.py' files in the appropriate directories on the server and client machines.

4.2 Running the Server

```
python server.py
```

4.3 Running the Client

```
python client.py
```

5 Known Issues

- The QKD simulation may not be fully accurate due to limitations in classical computing environments.
- The model decryption process may introduce some latency, depending on the size of the LLM and the computational power of the client machine.

6 Conclusion

This project successfully integrates cutting-edge cryptographic techniques with machine learning, offering a secure and innovative approach to deploying and utilizing LLMs in a networked environment. The use of QKD ensures that the system remains secure even in the face of future quantum computing threats.