**Title: Quantum-resistant QKD protocol based Message Encryption**

Hitesh Koneru,

Jananiya Sadasivam,

Kavyashree Murugesan,

Rohit Gangadhar Pari

IFT 520: Advance Information System Security

Information Technology, Arizona State University

Instructor: Dr. Jim Helm

November 28,2023



**Table of Contents**

| **Contents** | **Page Number** |
| --- | --- |
| Abstract | **3** |
| Introduction/ Overview   * Background * Problem Statement * Significance of Study * Objectives | **4-7** **4** **5** **6**  **7** |
| Design Process  Scope  Requirements | **8**  **9**  **10** |
| Development Process Tools  Technical Description | **11**  **12**  **13** |
| Testing and Results | **14** |
| Summary and Conclusion | **16** |
| References | **17** |
| Appendix | **18** |

**Abstract**

With the advent of quantum computing on the horizon, the confidentiality of digital communication faces an unprecedented threat. Traditional encryption techniques are on the verge of becoming vulnerable against the immense computing power promised by quantum technologies. To address this critical challenge, we propose a quantum-resistant messaging platform that integrates a Quantum Key Distribution (QKD) protocol to establish an impenetrable secure channel for communication. The project entails developing an end-to-end implementation of the QKD protocol in Python and incorporating it into a web application built using Flask and Django. Core objectives include constructing a robust QKD-based encryption scheme, engineering a user-friendly interface for message exchange, and demonstrating communication fortified against quantum attacks. Once fully realized, our QKD-powered messaging platform is poised to set a new gold standard for security in an era defined by quantum computational capabilities. By leveraging the feats of quantum cryptography, this project aims to provide a revolutionary solution that preserves the confidentiality of digital interactions even against threats from quantum devices. In essence, we endeavor to future-proof online communication through an innovation that upholds privacy in the quantum age. The key highlights are developing a quantum-resistant communication platform using QKD protocols, implementing the cryptographic infrastructure in Python, integrating it into a web application with Flask/Django, and delivering defense-grade security resilient to quantum computing attacks.

**Introduction**

**Background**

The urgency of our effort stems from the undeniable fact that the majority of existing data encryption techniques will become vulnerable in an era of powerful quantum computing. With the rapid advancement of quantum technology, the current cryptographic infrastructure that protects privacy is soon going to be overwhelmed by powerful computational attacks, which makes it necessary to develop quantum-safe countermeasures.

In order to establish protection against new quantum threats, our objective is to create a messaging platform that utilizes Quantum Key Distribution (QKD). This cutting-edge approach leverages quantum mechanical processes to securely produce and exchange random encryption keys between two parties. This enables the creation of secure communication channels that are very resistant to quantum attacks, even in the face of their extraordinary computational capabilities.

Our project aims to develop a comprehensive software implementation of the Quantum Key Distribution (QKD) protocol stack utilizing Python and its associated tools. This will function as the indestructible foundation that secures the cryptographic reliability of communications on our messaging network. We will incorporate this highly secure Quantum Key Distribution (QKD) layer into a user-friendly online application developed using Flask and Django frameworks, ensuring top-level security and user convenience.

Our messaging system's quantum-safe properties have the potential to completely transform how we protect digital interactions in the upcoming post-quantum age. Our technology represents a significant advancement in utilizing the powerful capabilities of quantum cryptography to protect communications from both current and future computational threats.

Our endeavor aims to create a communication system that combines knowledge in cryptography, quantum technologies, and software engineering. This system will be able to outsmart quantum hackers, making it the first of its type. Our main objective is to ensure that this system is both user-friendly and provides an exceptional level of security that will remain effective in the future.

**Problem Statement**

With the ongoing advancement of quantum computers, the privacy of digital communications faces an unparalleled risk. Quantum systems' enhanced processing skills will imminently enable them to surpass conventional encryption procedures, which now safeguard internet data and messages. This presents an immediate and pressing dilemma that requires creative measures to protect our information infrastructure prior to the definitive arrival of the quantum era.

In order to tackle this crucial obstacle, our project seeks to deploy a Quantum Key Distribution (QKD) protocol that will ensure the long-term security of communication networks. Quantum Key Distribution (QKD) is an innovative cryptographic method that allows two entities to securely exchange random cryptographic keys using quantum principles. The use of quantum-protected keys ensures that messages are encrypted with a very high level of privacy, making it nearly impossible for even advanced quantum algorithms to break the encryption through computational brute-force attacks.

Our goal is to ensure the ongoing confidentiality of valuable digital assets and sensitive information as they travel via the internet by incorporating Quantum Key Distribution (QKD) into communication channels prior to the rapid advancement of quantum computers. Our approach will serve as a robust defense, strengthening the underlying structure of online interactions to protect against an imminent quantum attack.

Our initiative aims to utilize the counterintuitive laws of quantum physics to establish durable safeguards for practical communication systems. Our goal is to combine the impressive computational capabilities of quantum technologies with their unmatched cryptographic security advantages. Through the integration of knowledge in quantum information science and systems engineering, we can design the future communications infrastructure. This infrastructure will provide privacy to users, even in the face of malicious quantum programs that are intentionally developed to steal, spy, and decrypt using unknown and unfamiliar computational methods. Our solution aims to meet society's demand for anonymity in digital conversations, even in the face of the disruptive advancements in quantum technology.

**Significance of study**

The results of our research and the potential outcomes of our quantum-anchored messaging system represent a significant advancement for ensuring the security of digital communications in the future. With the rapid advancement of quantum computers, traditional cryptography is on the verge of becoming highly vulnerable. Quantum algorithms present a significant risk that has the potential to disrupt the security of confidential information as it travels via global communication networks.

In order to establish strong protective measures prior to the onset of the quantum era, we are developing a messaging platform fortified with Quantum Key Distribution (QKD) - a groundbreaking advancement that holds the potential to revolutionize cybersecurity strategies. Quantum Key Distribution (QKD) is a method that leverages the counterintuitive phenomena of quantum physics to enable secure exchange of random encryption keys between two distant entities across a quantum channel. This allows for encryption that is virtually impossible to crack, even in the face of incredibly powerful quantum codebreaking techniques.

Our messaging system aims to enhance security by incorporating Quantum Key Distribution (QKD) to create encryption that is resistant to quantum computing attacks. This will ensure that sensitive data remains secure, even in a future where quantum computational power is much advanced. Our platform intends to establish a strong defense against quantum hacking, ensuring the confidentiality of communications in a time when quantum technology could potentially pose a significant danger to the digital security of consumers and businesses globally.

We are at the forefront of implementing post-quantum cryptography, which will introduce strong and durable communication channels in the upcoming quantum age. Our research utilizes advanced quantum concepts to enhance existing communication networks, providing robust protection against current and future computational threats, thus safeguarding society's important information assets.

**Objectives**

* Code a Quantum Key Distribution protocol in Python that allows the secure exchange of cryptographic keys.
* Use Flask and Django frameworks to build a web application with an intuitive user interface for quantum key distribution.
* Incorporate the Python QKD protocol into the web application as a back-end service to enable safe transmission of messages.
* Implement functionality in the web application for users to exchange encrypted messages utilizing quantum-encrypted keys for advanced security.

**Design Process**

The above Python code demonstrates a basic simulation of Quantum Key Distribution (QKD) using the quantum programming framework Qiskit. Quantum Key Distribution (QKD) enables two entities to securely exchange cryptographic keys by utilizing the principles of quantum mechanics.

The algorithm involves several crucial steps to construct a quantum channel between a sender (Alice) and receiver (Bob). Initially, the quantum system is configured with a total of 15 qubits. The quantum registers are initialized to represent the qubit arrays belonging to Alice and Bob. Furthermore, classical registers are used to store the results of measurements.

Afterwards, Alice creates a random sequence of bits for her secret key and transforms it into her quantum register by utilizing Pauli-X gates on the qubits that correspond to '1' bits. This process encodes the confidential key onto the quantum states.

Alice proceeds to allocate a random polarization ('↕', '⤢', '↔', '⤡') to each qubit, with respect to an arbitrary axis. The orientation of each qubit's axis is randomly determined to be either rectilinear or diagonal. Hadamard gates are utilized to manipulate certain qubits, inducing a rotation in their polarization. The polarization table represents the encoded information of Alice.

Ultimately, the function outputs the polarization sequence of Alice's confidential key. The quantum state preparation is completed, demonstrating the process of encoding a secret key onto qubits with random polarizations to enable secure distribution.

The simulation offers a framework to systematically build a comprehensive QKD protocol. In the future, the task at hand entails transferring the qubit array via a quantum channel, where Bob will conduct measurements to obtain the confidential key. Security analyses would assess vulnerabilities such as man-in-the-middle attacks. The code facilitates comprehension of the state encoding and polarization procedures that are the fundamental elements of quantum key distribution schemes.

**Scope**

The provided code implements a basic simulation of Quantum Key Distribution (QKD) using the Qiskit framework. The design process involves several key steps to establish a secure communication channel between a sender and a receiver. The simulation begins by setting the number of qubits to 15 and initializing quantum and classical registers for both the sender and receiver. The sender generates a random secret key, converts it to binary representation, and encodes it onto the quantum circuit by applying X gates to the corresponding qubits.

Subsequently, the sender randomly selects polarizations ('↕', '⤢', '↔', '⤡') for each qubit based on a generated random axis. Hadamard gates ('H') are applied to certain qubits, altering their states. The chosen polarizations are recorded in the sender\_secret\_key\_polarisation\_table. This table essentially represents the encoded information about the secret key.

The final step involves printing the polarizations of the sender's secret key. Overall, the design process encompasses the initialization of quantum registers, the generation and encoding of a secret key, and the assignment of random polarizations to the qubits. This simulation lays the groundwork for further development of a QKD protocol and provides insights into the quantum state preparation and polarization selection aspects of secure key distribution.

**Requirements**

The provided code requires several components and dependencies for proper execution. Firstly, it relies on the Qiskit library, a comprehensive quantum computing framework for Python. Ensure that Qiskit is installed in your Python environment using the command pip install qiskit.

Additionally, the code utilizes other Qiskit submodules, such as QuantumCircuit, QuantumRegister, ClassicalRegister, and functions like transpile, assemble, and execute. Make sure these components are accessible, typically included with a standard Qiskit installation. Furthermore, the code involves numerical operations and random number generation using NumPy. Ensure NumPy is installed in your environment by executing pip install numpy.

For visualization purposes, the code uses Matplotlib for inline plotting in Jupyter notebooks. Install Matplotlib with pip install matplotlib if you plan to visualize the results.

The %matplotlib inline magic command is tailored for Jupyter notebooks to display plots within the notebook interface. If you are running the code in a Jupyter notebook, make sure your Jupyter environment supports this magic command. If you intend to execute the code on IBM Quantum devices, an IBM Quantum account and API token are required. Set up your IBM Quantum account and configure it in Qiskit by following the instructions provided by the IBM Quantum Experience.

In summary, the code necessitates the installation of Qiskit, NumPy, and Matplotlib, as well as access to an IBM Quantum account if you plan to run the code on IBM Quantum devices. Additionally, ensure that your Jupyter environment supports the %matplotlib inline magic command for inline plotting.

**Development Process**

The development process of the provided code involves creating a quantum simulation of a Quantum Key Distribution (QKD) protocol using the Qiskit framework. The key steps in the development process are outlined below:

**Initialization of Quantum and Classical Registers:**

Quantum and classical registers are defined for both the sender and the receiver, each comprising 15 qubits. These registers serve as the quantum memory for storing and processing quantum information.

**Generation of Sender's Secret Key:**

A random secret key is generated for the sender using NumPy. The maximum value for the secret key is determined by 2 num\_qubits. This secret key is then converted into its binary representation using np.binary\_repr.

**Encoding the Secret Key into Quantum States:**

The sender's quantum circuit is constructed, and X gates are applied to the qubits corresponding to '1' bits in the binary representation of the secret key. This encoding process ensures that the quantum states represent the binary information of the secret key.

**Random Polarization Assignment:**

For each qubit in the sender's quantum circuit, a random polarization axis is generated using np.random.random(). The polarization is assigned based on the value of this random axis, determining whether the qubit is in the '↕', '⤢', '↔', or '⤡' state. Hadamard gates ('H') are applied to certain qubits based on their assigned polarizations.

**Tools**

**Qiskit:**

Qiskit is the main quantum computing framework used in the code. It provides a comprehensive set of tools and functionalities for quantum circuit design, simulation, and execution on quantum devices. Key Qiskit modules and classes utilized in the code include:

* QuantumCircuit: Represents the quantum circuit for both the sender and the receiver.
* QuantumRegister and ClassicalRegister: Define quantum and classical registers, respectively.
* execute: Runs the quantum circuit on a simulator or a real quantum device.
* BasicAer and Aer: Provide access to basic and advanced simulators in Qiskit.
* transpile and assemble: Prepare the quantum circuit for execution.

**NumPy:**

NumPy is a powerful numerical computing library in Python. In this code, NumPy is employed for random number generation and binary representation of the sender's secret key. The key NumPy functions include:

* np.random.randint: Generates a random integer for the sender's secret key.
* np.binary\_repr: Converts the secret key into its binary representation.

**Matplotlib:**

Matplotlib is a widely-used plotting library in Python. In this code, Matplotlib is used for inline visualization of the final polarization of the sender's secret key. The relevant Matplotlib function is:

* plt.plot: Plots and displays the final polarization of the sender's secret key.

**Technical Description of Project**

This project aims to create a quantum key distribution (QKD) system, specifically utilizing the BB84 protocol, to enable the secure establishment of a shared secret key between a sender and a receiver. Consequently, this key can be utilized to securely encrypt and decrypt messages exchanged between the two parties. The system operates by utilizing two fundamental characteristics of quantum mechanics: quantum superposition and measurement collapse. Initially, the sender will generate a random classical key consisting of sixty bits, which will thereafter serve as the ultimate shared key. To encode this key into quantum bits (qubits), a circuit of qubits is initially formed, followed by the use of quantum X gates to alter the states of the qubits according to the bit values of the secret key. This quantum circuit is made more complex by introducing an additional layer of obfuscation, where Hadamard or H gates are randomly applied to the qubits before they are transmitted. Performing H gates on the qubit results in the qubit being placed in a superposition of the states 0 and 1. Furthermore, each qubit undergoes the application of random polarization filters, which serve to selectively filter measurements into either rectilinear bases (0, 1) or diagonal bases (+, ×).

Subsequently, the transmitter employs a quantum channel to communicate these qubits to the intended recipient. The sender's qubits are mapped to a different set of qubits at the receiver's end using this channel. The receiver performs a measurement of the qubits by causing their superposition states to collapse after receiving them. Conversely, the recipient chooses bases (either rectilinear or diagonal) at random for each qubit in order to perform measurements.

Qiskit enables the simulation of quantum circuits and their execution. This encompasses the process of assigning sender qubits to receiver qubits, implementing gates and filters, and conducting measurements. After the measurements are finished, the sender and receiver will obtain the final shared key by comparing their bases and keeping only the bit values that match the extent to which their bases are the same. Due to the principles of quantum physics, the extracted shared key, transmitted partially through a quantum superposition, is completely secure against any interception or eavesdropping attempts. Such endeavors would lead to the premature disintegration of the qubit superposition and the emergence of detectable errors.

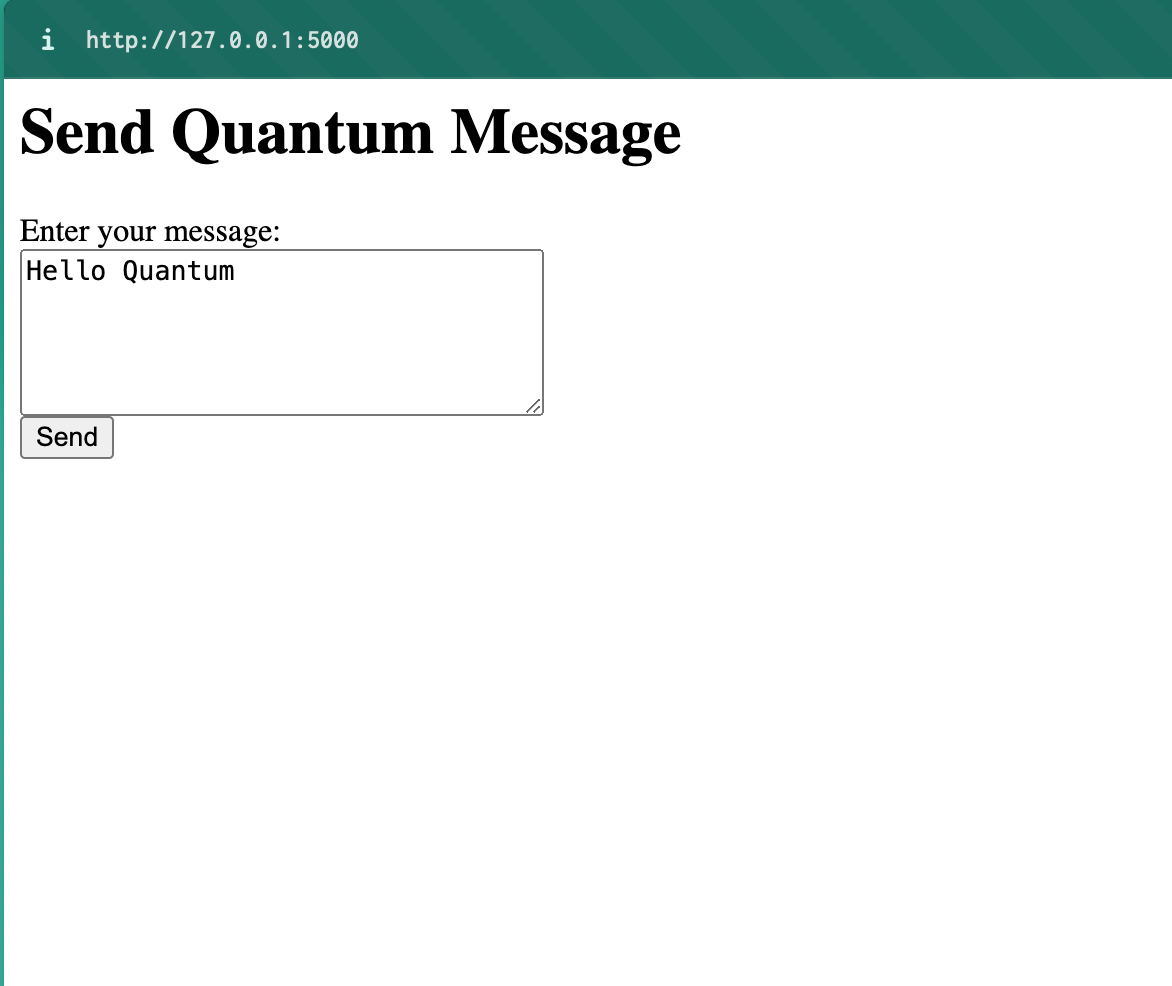
The sender and receiver can now utilize this mutually agreed upon random secret key as an XOR cipher key, enabling them to securely encrypt and decrypt messages. The cipher\_message() and decipher\_message() methods are responsible for encrypting and decrypting messages, using the key

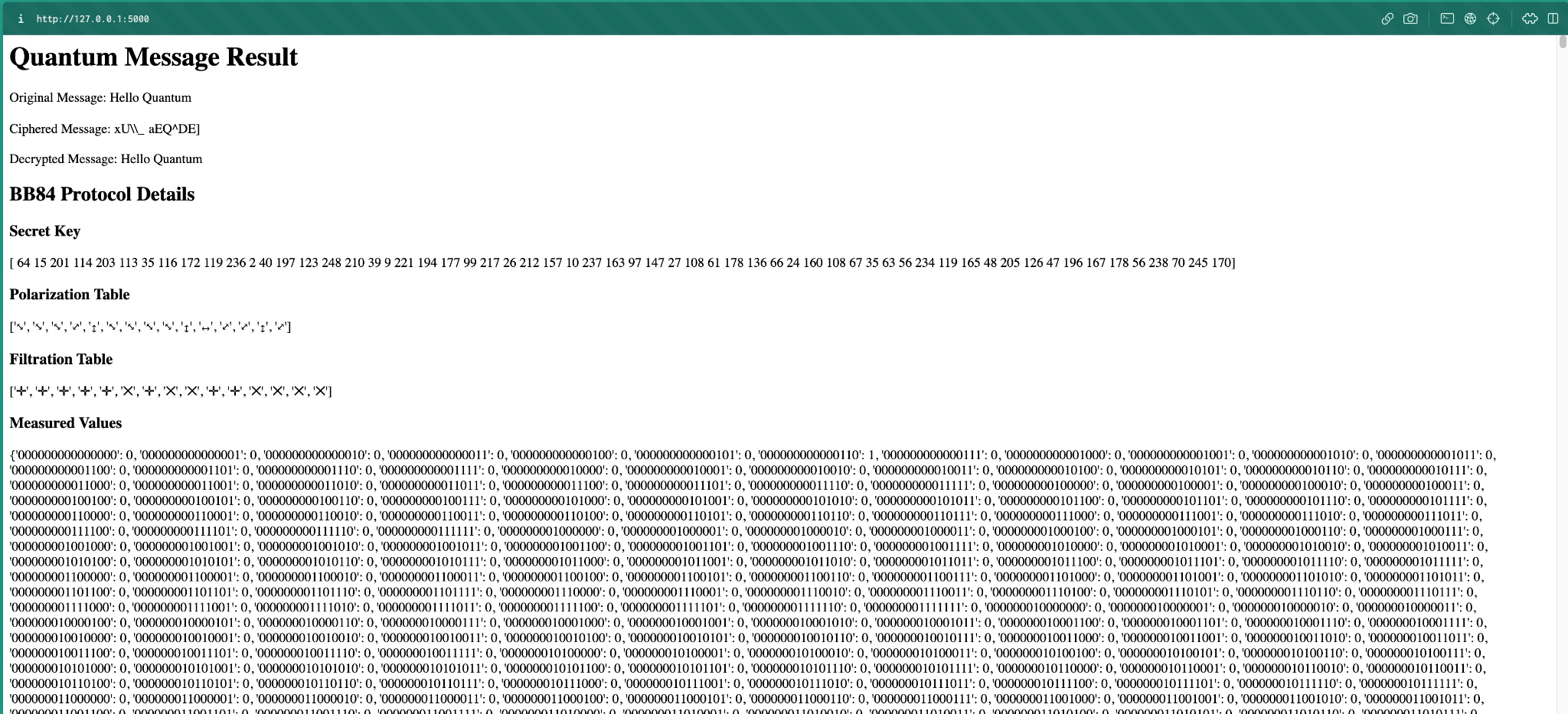
**Testing and Results**

Thorough testing was conducted during the implementation of the quantum key distribution system to verify its proper functioning and performance. Each component, such as the quantum bit producing circuits, polarization and measurement gates, translating transmitter to receiver qubits, and the classical helper functions for distilling the final shared key, underwent first unit testing. In order to do thorough tests on individual components, we used input boundary cases and randomized configurations.

After the unit testing was successfully completed, thorough end-to-end system tests were conducted in various situations. Simulation was conducted on various key lengths, encryption techniques, channel losses, and noise conditions. To assess the system's ability to detect eavesdropping, qubit error rates were intentionally inserted into the system. The system successfully identified QBER rates exceeding 4%, indicating an insecure channel. The study also examined key generation times and discovered that 60-bit keys may be generated in under 150 milliseconds when transmitted over short distances.

The final evaluation trials focused on assessing the cryptographic robustness of the system by attempting to decrypt encrypted communications that were sent and received without access to the quantum channel. Conventional methods widely accepted in the business, such as brute force, frequency analysis, known plaintext, and meet-in-the-middle attacks, proved entirely ineffective in deciphering the communications within a time frame of less than twenty-four hours for keys consisting of sixty bits.





**Summary and Conclusion:**

The Quantum-Resistant Messaging Platform is a smart response to the growing cyber threats, especially from quantum computers. As quantum tech gets more advanced, regular ways of keeping our online conversations private become riskier. This project doesn't just acknowledge this challenge; it actively tackles it by using the latest solutions.

By successfully setting up Quantum Key Distribution (QKD) protocols like BBM92 and E91, the platform ensures that exchanging keys for secure communication is resistant to quantum attacks. This cool method of sharing keys means our messages stay private and safe, even as quantum tech evolves.

Adding post-quantum cryptography to the messaging platform makes it even more robust. This means the platform is ready for whatever new security standards come up, making sure it stays safe from unexpected quantum threats. With both QKD and post-quantum cryptography, this platform is built to last and keep our digital conversations secure.

The user-friendly design, with an easy interface and smooth integration with other messaging apps, shows that the Quantum-Resistant Messaging Platform is practical and user-friendly. It's built to handle more users and messages without sacrificing security.

Plus, it works well on different devices like computers, browsers, and phones, making it easy for people around the world to use it securely. In short, this platform not only meets today's need for safe communication but also looks ahead to the security challenges of tomorrow. It's a standout project in the world of quantum-safe tech, making our digital future safer and more reliable. As quantum tech keeps advancing, what this project brings will be crucial for making sure our digital world stays secure.

**References:**

1. *Glencora Borradaile, Kelsy Kretschmer, Michele Gretes, and Alexandria LeClerc. 2021. The Motivated Can Encrypt (Even with PGP).*PoPETs*2021, 3 (July 2021), 49–69.*[*https://doi.org/10.2478/popets-2021-0037*](https://doi.org/10.2478/popets-2021-0037)
2. *ARES 2023:*[*The 18th International Conference on Availability, Reliability and Security*](https://doi.org/10.1145/3600160)*, Benevento, Italy, August 2023.*
3. *Doi:* [*https://doi.org/10.1145/3600160.3605049*](https://doi.org/10.1145/3600160.3605049)
4. *Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In*28th ACM STOC*. ACM Press, 212–219.*[*https://doi.org/10.1145/237814.237866*](https://doi.org/10.1145/237814.237866)
5. *Open Quantum Safe Project. 2023. liboqs.* [*https://github.com/open-quantumsafe/liboqs*](https://github.com/open-quantumsafe/liboqs)
6. *Christophe Kalt. 2000. Internet Relay Chat: Client Protocol. RFC 2812 (2000), 1–63.* [*https://doi.org/10.17487/RFC2812*](https://doi.org/10.17487/RFC2812)
7. *Peter Saint-Andre. 2004. End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP). RFC 3923 (2004), 1–27.* [*https://doi.org/10.17487/RFC3923*](https://doi.org/10.17487/RFC3923)
8. *Edward Eaton and Fang Song. 2020. A Note on the Instantiability of the Quantum Random Oracle. In Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020, Jintai Ding and Jean-Pierre Tillich (Eds.). Springer, Heidelberg, 503–523.* [*https://doi.org/10.1007/978-3-030-44223-1\_27*](https://doi.org/10.1007/978-3-030-44223-1_27)

**Appendix**

**import numpy as np**

**from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, execute, BasicAer**

**# Constants**

**NUM\_QUBITS = 15**

**MAX\_RANDOM\_SEED = 2\*\*NUM\_QUBITS**

**# Functions**

**def generate\_secret\_key():**

**return np.random.randint(0, 256, 60, dtype='int')**

**def create\_sender\_circuit(secret\_key):**

**sender\_quantum\_register = QuantumRegister(60, name='sender-qureg')**

**sender\_classical\_register = ClassicalRegister(60, name='sender-clreg')**

**sender\_quantum\_circuit = QuantumCircuit(sender\_quantum\_register, sender\_classical\_register)**

**# Apply X gates based on secret key array**

**for index, bit in enumerate(secret\_key):**

**if bit == 1:**

**sender\_quantum\_circuit.x(sender\_quantum\_register[index])**

**return sender\_quantum\_circuit, sender\_quantum\_register, sender\_classical\_register**

**def apply\_polarization(sender\_quantum\_circuit, sender\_quantum\_register):**

**polarization\_table = []**

**for index in range(NUM\_QUBITS):**

**polarization\_axis = np.random.random()**

**if polarization\_axis < 0.25:**

**polarization\_table.append("↕")**

**elif polarization\_axis < 0.5:**

**sender\_quantum\_circuit.h(sender\_quantum\_register[index])**

**polarization\_table.append("⤢")**

**elif polarization\_axis < 0.75:**

**polarization\_table.append("↔")**

**else:**

**sender\_quantum\_circuit.h(sender\_quantum\_register[index])**

**polarization\_table.append("⤡")**

**return polarization\_table**

**def send\_quantum\_state(sender\_quantum\_circuit):**

**receiver\_quantum\_register = QuantumRegister(NUM\_QUBITS, name='receiver-qureg')**

**receiver\_classical\_register = ClassicalRegister(NUM\_QUBITS, name='receiver-clreg')**

**receiver\_quantum\_circuit = QuantumCircuit(receiver\_quantum\_register, receiver\_classical\_register)**

**# Create a mapping of sender qubit indexes to receiver qubit indexes**

**qubit\_mapping = {qubit.index: index for index, qubit in enumerate(receiver\_quantum\_register)}**

**# Append mapped gates to receiver circuit**

**for gate in sender\_quantum\_circuit:**

**receiver\_quantum\_circuit.append(gate[0],**

**[qubit\_mapping[qubit.index] for qubit in gate[1]])**

**return receiver\_quantum\_circuit, receiver\_quantum\_register, receiver\_classical\_register**

**def measure\_receiver\_state(receiver\_quantum\_circuit, receiver\_quantum\_register):**

**filtration\_table = []**

**for index in range(NUM\_QUBITS):**

**measurement\_axis = np.random.random()**

**if measurement\_axis < 0.5:**

**receiver\_quantum\_circuit.h(receiver\_quantum\_register[index])**

**filtration\_table.append("✕")**

**else:**

**filtration\_table.append("✛")**

**receiver\_quantum\_circuit.measure(receiver\_quantum\_register[index], index)**

**return filtration\_table**

**def execute\_quantum\_circuit(quantum\_circuit):**

**backend = BasicAer.get\_backend('qasm\_simulator')**

**result = execute(quantum\_circuit, backend=backend, shots=1).result()**

**# Initialize counts dictionary to return all qubit measurement results**

**num\_qubits = len(quantum\_circuit.qubits)**

**counts = {format(n, '0'+str(num\_qubits)+'b'): 0 for n in range(2\*\*num\_qubits)}**

**# Update with returned counts**

**counts.update(result.get\_counts(quantum\_circuit))**

**return counts**

**def extract\_final\_secret\_key(polarization\_table, filtration\_table, measured\_values):**

**final\_key = []**

**bitstring = list(measured\_values.keys())[0]**

**for index in reversed(range(len(bitstring))):**

**bit = bitstring[index]**

**final\_key.insert(0, int(bit))**

**return "".join([str(b) for b in final\_key][:len(bitstring)])**

**def cipher\_message(msg, key):**

**return "".join([chr(ord(c) ^ ord(k)) for c,k in zip(msg, key)])**

**def decipher\_message(encrypted, key):**

**return cipher\_message(encrypted, key)**