Final Report

Post-Quantum Cryptography

L3Harris

Team Member Names :
Aidan Heffron
Josef Munduchirakal
Blake Lun
Grant Shields
Shifan Hirani

CSCE 482 – Senior Capstone Design
Spring 2024

Texas A&M University

Department of Computer Science and Engineering

Contents

1. Abstract

As we navigate the uncharted waters of quantum computing, the urgency of implementing post-quantum cryptography becomes paramount. Many applications today rely on classical cryptography algorithms, algorithms that are not resistant to quantum computing, to secure millions and millions of private data for millions of different applications. Every day, quantum computing advances in ways previously unimaginable, making the threat of quantum decryption more and more relevant.

A number of resources are out there to help application and web owners implement quantum-safe protocols, cryptography protocols that are resistant to quantum computing. However, certain applications such as OpenZiti (a zero-trust network provider) rely on incompatible source code to these different resources. Moreover, many web owners cannot determine if the trade-off of the increased security is worth the increased demand of providing quantum-safe cryptography.

Our solution is to create the relevant wrappers that enable quantum-safe X.509 protocols for more applications. With this completed, we will develop our own post-quantum tunnel between a server and client application. This will allow us to conduct extensive performance testing to track exactly what the trade-off is as you switch from classical cryptography to quantum-safe cryptography. This information will be useful for any admin looking into providing a more secure service and can help them understand how their clients will be affected.

Our solution will be tested against complex network topologies, simulated around the nation. The novelty of our performance testing will be used to provide graphics and information to anyone interested in the trade-off of classical and post-quantum cryptography. With this, network administrators can directly compare classical and post-quantum cryptography, to see things such as how post-quantum certificates and keys are more than six times the size of some of their classical counterparts (RSA certificate is roughly 1000 bytes, Dilithium3 certificate is roughly 7000 bytes), or that in certain network conditions (like noisy connections), post-quantum communication can take up to or beyond twice as long as classical (under high loss, sending a post-quantum Dilithium3 certificate takes 103 microseconds, where as an RSA certificate takes 36 microseconds).

2. Introduction

2.1 General Scope and Problem Background

2.1.1 Motivation

Cryptography is one of the most important concepts in computer science and mathematics ever developed. It is so prominent that after World War II, it became illegal to sell or distribute many different cryptographic information overseas. The sheer impact on what would happen if what was believed to be a secure network that got hacked can be very devastating.

Currently, modern computers and networks utilize *classical cryptography*, which are algorithms that are not resistant to quantum computers. *Post-quantum cryptography* refers to cryptographic algorithms that are believed to be resistant to quantum computers.

Quantum computing offers attacks on cryptography never before believed to be possible. Algorithms such as Shor's algorithm can be used to completely break down classical encryption methods such as RSA, one of the most commonly used encryption algorithms in cryptography today. So many applications rely on cryptography, from banking to military to web servers, etc. We want to help ease the implementation of post-quantum cryptography into these applications and moreover study the impact on what this does in terms of performance.

There are a few libraries that exist already to provide post-quantum cryptography, the most well known and trusted being LibOQS (Open-Quantum Safe). This library can be implemented into almost any language and used to go through the steps of key encapsulation and signing with post-quantum algorithms. It also includes basic performance measuring capabilities for this process to help show the impact on what it offers.

LibOQS however, does not provide any language wrappers for the X.509 implementation. The generation of post-quantum certificates, parsers, and verification of these certificates must happen outside of any SDK an application might be using. Any server that wishes to use post-quantum X.509 to verify their clients must implement their own wrapper to do so. Along with this, there is no way to measure information about the packets that flow between the client and server or how the system would react in different topologies with different numbers of clients. This information is critical for web servers as they need to know if the implementation will affect what systems their clients can use, how long it may delay data transfer, etc.

2.1.2 Domain Context

Throughout this report, we will be referencing different open source softwares such as LibOQS, OQS-Provider, and OpenSSL. Together, these different softwares will offer different functionalities that we will be leveraging to expand the current solution into new, uncharted areas. We will be using wireshark and Mininet to capture and simulate network topologies pertinent to us.

According to Forbes, over 90% of the internet relies on RSA for initiating SSL handshakes. This means that when quantum computing becomes more mainstream, so many applications hosted today will be at risk of full on attacks from outside sources.

One of these applications is OpenZiti, an open source, zero-trust network provider bent on classical X.509 and encryption with RSA. One of the struggles with implementing post-quantum X.509 within OpenZiti is that ziti is written in Go, a language that is incompatible with the OQS X.509 provider.

Classical cryptography implies encryption through conventional methods such as RSA. These methods of encryption have been around for years. However, they are vulnerable to the brute force of quantum computers. Post-quantum encryption is a more novel form of encryption which attempts to have increased resistance to the computational power of quantum computers. An important note regarding the difference between classical and post-quantum algorithms is that the computational resources to run each algorithm varies. Post-quantum algorithms in some cases may run faster but will almost always have a larger file size than classical algorithms. Our report will attempt to illustrate each one of these tradeoffs in detail.

2.2 Goals and Objectives

With this project, we will explore a number of different objectives, a full list of everything provided in the Project Plan (Annex 1).

The scope of this project is to:

- Implement a Go library for OQS X.509
  - Certificate Generation
  - Certificate Verification
  - Certificate Parsing
- Import the library into an original client/server tunnel application
- Leverage the library ensure quantum-safe authentication and encryption between the client and server
- Implement a performance suite to time the different elements of cryptography and measure the difference between performance of the classical algorithms (like RSA)
- Use Mininet to simulate our client/server application in a number of different configurations with a number of different nodes

Following the above scope will lead us to the following deliverables:

- A fully functional, importable Go library that offers OQS X.509
- Graphics illustrating the time differences between quantum-safe and classical encryption
- Graphics illustrating the performance requirements of quantum-safe cryptography
- Wireshark captures proving the trade-off between security and performance for quantum-safe cryptography elements
- An exportable Mininet program that simulates different topologies

2.3 Design Constraints

All of the software we are working with is open source. This means that anyone should be able to view the source code and implement it completely for free into their programs. Our solution needs to follow this precedent. We should not be implementing any restrictive licenses into our code that would jeopardize the integrity or availability of the open source software. We should not be using anything that requires a paid subscription or using anything that would require people who consume our library to be at risk of any type of audit. This way, our solution will be truly open and free to be used by anyone.

Moreover, the technology required to run the different algorithms acts as a constraint. These post-quantum algorithms are highly sophisticated and can sometimes require computing power provided in your modern day laptop. However, if you were to run the same algorithms on an older piece of hardware, the time difference would be noticeable

2.4 Solution

Our solution consists of three separate parts: the Go Library, the client/server tunnel, and the performance suite.

The Go Library:

- This library addresses the problem of there not being a Go implementation of PQ X.509 protocols for certificate generation and verification.
- Softwares such as OpenZiti would be greatly aided by a library they could just import into their code base that does all they need in terms of certificates.
- This library will provide Go functions that can be called to generate a certificate from an approved signing algorithm, verify a certificate given a key, and parse a certificate to extract relevant information (such as issuer, expiration date, etc.)

The Client/Server Tunnel:

- A lot of the novelty in our solution comes from the performance suite, something that cannot be done without an application to test on
- This tunnel application will consist of three different components, the client/server authentication with PQ X.509 certificates (created from our Go library), post-quantum Key Exchange Mechanism, and AES256 encryption of messages.
- Together, all three of these components work together to create a fully secure, post-quantum tunnel that allows a client to communicate with a server

The Performance Suite:

- This suite will be used to measure a ton of different data points throughout the solution
- Timer code will be implemented to measure and average the time it takes for different computers to generate and verify post-quantum certificates and keys

- Mininet will be used to simulate the client speaking to the server across different topologies to show how adding latency may compound the time it takes for a client to authenticate and talk with a server
- Mininet will also be used to simulate multiple clients connecting to a server at the same time to determine performance requirements for a server that is issuing and verifying post-quantum certificates

2.5 Evaluation Plan

To evaluate whether or not this solution meets the agreed-to requirements, we will be conducting a number of different tests. Most importantly, everything needs to be able to compile and work together. A Go library that cannot be imported into other programs is not an acceptable Go library. Unit testing and integration testing of each individual component of both the library and the tunnel application will be conducted to ensure that our code does as it says it does.

Moreover, a critical component of our evaluation plan is the User Acceptance Testing. Putting our solution in the hands of different stakeholders will allow them to see for themselves exactly what our programs do and that they accomplish what they were set to accomplish. We will be able to get relevant feedback on how to improve our solution and scale it.

The rest of this report dives much deeper into our solution, from its design to its testing and results. We will explore deeper into related work and how we are providing a novel solution.

3. Related Work

Throughout this report, we will be referencing and using many different sources to motivate different ideas related to cryptography and networking. These will mostly stem from the following articles and sources, of which we have prepared summaries to help provide a basic understanding of what we have researched.

Source:

Bos, J., Ducas, L., & Kiltz, E. (n.d.). Crystals - Kyber: A CCA-secure module-lattice-based Kem | IEEE ... https://ieeexplore.ieee.org/abstract/document/8406610

The article discusses the rise of interest in post-quantum cryptographic schemes due to advancements in quantum computing and the need for new standards set by organizations like NIST. It focuses on lattice-based cryptography, particularly the use of lattice problems' hardness as a basis for cryptographic constructions.

Initially, lattice cryptography attracted attention for its security proofs based on worst-case instances of lattice problems. It should be noted that preparation for worst-case scenarios is not the only measure of a secure algorithm. The Learning With Errors (LWE) problem emerged as a crucial component in lattice-based constructions, offering simplicity and hardness similar to standard lattice problems. The article introduces the Ring-LWE assumption, demonstrating its equivalence to LWE over certain polynomial rings. This assumption enables the creation of efficient encryption schemes. Much of the article is spent explaining the algorithms and proofs required to define a secure network. Different parameter settings lead to variations like Ring-LWE, LWE-based, or Module-LWE schemes, each with trade-offs in efficiency and susceptibility to attacks.

The paper presents Crystals-Kyber, a highly optimized CCA-secure KEM (Key Encapsulation Mechanism) based on the hardness of Module-LWE. It discusses Kyber's flexibility, security advantages, and performance, highlighting its efficiency comparable to Ring-LWE-based schemes. The paper concludes with the effectiveness of Kyber in terms of its performance in cycles.

Source:

Debnath, S. K., Choudhury, T., Kundu, N., & Dey, K. (2021, January 29). Post-quantum secure multi-party private set-intersection in Star Network topology. Journal of Information Security and Applications. https://www.sciencedirect.com/science/article/abs/pii/S2214212620308668

The paper details the design of the Multi-Party PSI (MPSI) protocol using a star network topology, where a designated party communicated individually with others to compute the intersection securely. A PSI is a Private Set Intersection, which is a technique for securely computing set operations on private data sets. The paper provides an example of how data might be shared, and ideally, not share more than necessary. Consider two law enforcement agencies wanting to determine overlap in their list of suspects, without sending their respective lists over in their entirety. Having a third party to compare would be nice, but unrealistic for large-scale replication. Here, the PSI solution would allow parties to determine the intersection of their data sets privately.

While existing MPSI protocols rely on number theory assumptions, they face insecurity with the advent of quantum computing as it becomes more commonplace in the future. The MPSI proposed by the paper uses lattice-based encryption and Bloom filters. Bloom filters use hash functions to determine if an element exists within a set. The security of the protocol is analyzed using semi-honest adversaries in the standard model. Semi-honest implies that we are not dealing with hackers attempting to crack the encryption, but rather listening parties intending to gain information.

Source:

Kampanakis, P., Panburana, P., Daw, E., & Geest, D.V. (2018). The Viability of Post-quantum X.509 Certificates. *IACR Cryptol. ePrint Arch., 2018*, 63. https://eprint.iacr.org/2018/063.pdf

One of the most widely accepted cryptography standards is X.509. In this system, authentication certificates are exchanged between different hosts in order to authenticate a user to a server. X.509 relies heavily on public key generation, most often creating using RSA encryption, an encryption method that could very easily be broken by a quantum computer using any quantum decryption algorithm. Using a post-quantum implementation would surely increase the overhead of generating such certificates, yet as found in testing with OpenSSL and StrongSwan post-quantum systems, it is a viable solution.

This source is a very important direct reference as it notes that other systems in which operate a post-quantum certification system are still viable for applications. One of the biggest concerns with quantum keys is the size required of the key to sufficiently protect against quantum algorithms. This is data that we wish to expand upon as a team. We want to figure out whether the viability of quantum-safe cryptography changes as network topologies change. When there is a noisy tunnel between hosts and retransmission becomes frequent, does this affect whether or not the increased size in packets is worth it? When the network switches between nodes have limited bandwidth, is it worth the fragmentation? These are questions we want to solve when implementing our system.

Source:

R. L. S. de Oliveira, C. M. Schweitzer, A. A. Shinoda and Ligia Rodrigues Prete, "Using Mininet for emulation and prototyping Software-Defined Networks," 2014 IEEE Colombian Conference on Communications and Computing (COLCOM), Bogota, Colombia, 2014, pp. 1-6, doi: 10.1109. https://ieeexplore.ieee.org/abstract/document/6860404

Throughout the world, there are so many different network topologies that exist between clients and applications. These different topologies can play so many different roles on the performance and requirements of the hosts interacting within the network. For this reason, it is very important for application owners to understand how different topologies may affect the clients they are working with. Mininet is a virtualization software that allows for the simulation and testing of different network topologies, ones that are specific to so many different applications today.

This article is super important as it provides insight into the development of simulated topologies that our group will be using as we move forwards with our solution. Quantum cryptography on its own is an emerging technology that increases the size of public keys, private keys, X.509 certificates, etc. But how exactly does this increased size affect the networks in which the cryptographic components are sent? What happens when switches are restricted to low bandwidths and packets need to be fragmented before getting sent across the network? We will be exploring these questions and more, testing our solution in a number of different topologies all aimed at real world scenarios that many application owners may find interesting.

Source:
Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J., Schwabe, P., Seiler, G., & Stehlé, D. (2021, August 4). *Kyber-specification-ROUND3-20210804.PDF*. pq-crystals. https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf

Encryption methods such as RSA have been around for a while. These algorithms utilize a randomization technique to generate a key that is very difficult to guess through brute force. Whilst this method of encryption is very common today among numerous technological applications, it has certain limitations and vulnerabilities. The rise of quantum computers poses a threat to data that is encrypted via this traditional form of encryption. The CRYSTALS-Kyber algorithm is used in order to provide an encryption option that has the power to hold up the the computing power of quantum computers.

The National Institute of Standards and Technology (NIST) is the industry standard setting organization for the purposes of providing guidance on well tested encryption algorithms. The CRYSTALS-Kyber algorithm has proven to be effective in the NIST testing format and it has been chosen as one of the finalists in the standardization process. The key and ciphertext lengths are an important factor when it comes to choosing a quantum algorithm. One benefit of the CRYSALS-Kyber algorithm is that the size has been improved to only take up around half as much space. Because we would like to use a trusted post quantum encryption algorithm implementation within our project, the CRYSALS-Kyber algorithm is a great starting point.

Source:
Encrypt, J. A. L., Aas, J., Encrypt, L., Cisco, R. B., Barnes, R., Cisco, University, B. C. S., Case, B., University, S., University, Z. D. S., Durumeric, Z., Foundation, P. E. E. F., Eckersley, P., Foundation, E. F., University, A. F.-L. S., Flores-López, A., J. Alex Halderman

University of Michigan, Halderman, J. A., Michigan, U. of, … Metrics, O. M. A. (2019, November 1). Let's encrypt: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. ACM Conferences. https://dl.acm.org/doi/abs/10.1145/3319535.3363192

Currently the vast majority of sites on the web have encryption standards implemented but this has not always been the case. In times like the early versions of the web, many sites did not use proper encryption or did not adhere to general industry standards. One way in which the promotion of encryption within websites was able to be promoted was through Let's Encrypt. Let's Encrypt serves to be an automated certificate authority that is open for any site to take advantage of. Certificate authorities function like a verifier of identity. For example, if a user types in bank of america in their browser's search engine and clicks the first link, that user needs to know if the server that information is being sent to is the real bank of america server or just a poser. One way to do this is through certificate authorities which sign certificates and verify that the server is actually who they say they are.

In a short amount of time, Let's Encrypt has grown to be one of the largest Certificate Authorities by volume of certificates generated. This process of automation is similar to what we hope to use in our project. We would like to self sign certificates and have a local reference to the certificate authorities on our local computers. One of the primary steps of our project attempts to replicate this effort on a much smaller scale.

Source:
Dam, D. T., Tran, T. H., Hoang, V. P., Pham, C. K., & Hoang, T. T. (2023). A survey of post-quantum cryptography: Start of a new race. *Cryptography*, *7*(3), 40. https://www.mdpi.com/2410-387X/7/3/40

The transition towards Post-Quantum Cryptography (PQC) marks a significant phase in securing digital communications against the potential threats posed by quantum computing. The implications of quantum computing for cryptography are profound because many essential digital security protocols, including those used for secure communication (such as TLS/SSL), digital signatures, and encryption, rely on these cryptographic primitives. Foundational understanding of the current landscape of post quantum cryptography including the various approaches being considered to protect against the potential capabilities of quantum computers is essential to future proofing for any system.

The authors delve into the evolving landscape of cryptographic practices in the face of quantum advancements. The paper methodically examines the
vulnerabilities of classical cryptographic schemes to quantum attacks and highlights the progress in developing quantum resistant cryptographic algorithms. The authors provide a detailed overview of the current state of PQC, categorizing and comparing various quantum resistant algorithms that promise to safeguard future communications.Their survey includes a detailed examination of the various cryptographic algorithms proposed to counter quantum threats, such

as lattice based, hash based, code based, and multivariate polynomial algorithms. It particularly focuses on the efforts led by organizations such as the National Institute of Standards and Technology (NIST) in standardizing PQC algorithms which are the basis of the implementation of our project.

Source:
Moody, D. (2023, September 27). *CSRC presentation: And then there were Four: The first NIST PQC standards*. National Institute of Standards and Technology. https://csrc.nist.gov/presentations/2023/mpts2023-day2-talk-nist-pqc-first-standards

The realm of cryptography is on the cusp of a transformative shift with the advent of quantum computing. It necessitates the development of quantum-resistant cryptographic standards. This is where NIST comes in. NIST has taken foundational steps in establishing the first-standards for post-quantum cryptography.

In his presentation, Moody acknowledges the extensive body of research and collaborative efforts that have paved the way for the establishment of the first NIST PQC standards. He highlights the global involvement of academic experts, industry leaders, and governmental agencies in contributing to the research and development of quantum-resistant cryptographic algorithms. NIST's existing public-key crypto standards (SP 800-56A/B and FIPS 186) are vulnerable to quantum computer attacks. Symmetric-key cryptography (like AES and SHA) is less affected but still vulnerable.

The NIST initiated a call for quantum-resistant cryptographic algorithms to establish new public-key crypto standards, focusing on digital signatures and encryption/key establishment. The presentation talks about CRYSTALS-Kyber, which we are looking into heavily as part of our project. The presentation notes that all operations within CRYSTALS-Kyber are performed over a specific algebraic structure, utilizing elements from a Gaussian distribution. This implies that it leverages the hardness of lattice-based problems, which are supposed to be difficult for both classical and quantum computers to solve efficiently.

Source:
Stebila, Douglas, and Michele Mosca. "Post-Quantum Key Exchange for the Internet and the Open Quantum Safe Project." *SpringerLink*, Springer International Publishing, 20 Oct. 2017, link.springer.com/chapter/10.1007/978-3-319-69453-5_2#Sec14.

With the rise of quantum computers, concerns about future security from cryptographic algorithms have been surfacing. This paper explores the open quantum safe project. This project aims to facilitate the usage of quantum cryptography. The main component that we are interested in is the library called liboqs. This is an open source C library that provides implementations of various post-quantum cryptographic algorithms. By obtaining these cryptographic algorithms, we are able to implement their uses into our key and certificate generation to increase the level of security between users.

The OQS project enables organizations to prototype quantum algorithms in their applications. This promotes flexibility and adaptability when trying to handle improvements and changes in post quantum algorithms without significantly having to modify pre-existing software. By providing a common interface for key exchange and digital signature schemes, liboqs streamlines the integration process and facilitates runtime comparisons through its benchmarking program. This approach empowers organizations to explore quantum-safe options allowing them to navigate the complex intricacies of cryptographic algorithm selection and implementation with efficiency.

Source:
Bindel, N., Braun, J., Gladiator, L., Stöckert, T., & Wirth, J. (2019, August 12). X.509-Compliant Hybrid Certificates for the Post-Quantum Transition. Open Journals. https://www.theoj.org/joss-papers/joss.01606/10.21105.joss.01606.pdf

Quantum computers pose a big threat to classical cryptography since it exceeds the limits that are bounded by standard means. This paper introduces a java version of hybrid certificates that utilizes X.509 measures to showcase the effects on classical cryptography as well as post quantum cryptography. By combining the benefits of both classical and quantum cryptography, the new hybrid certificates offer a more secure approach to allow for the simple transition into security from quantum computing. The implementation of such cryptographic measures allows for testing and evaluation in real-world environments. This helps researchers and practitioners in optimizing the parameter setting for new cryptographic algorithms.

TLS certificates contain digital signatures, keys, and algorithms. Hybrid TLS certificates include extra X.509 certificate fields for quantum-safe keys and signatures, and the encoding for a quantum safe algorithm. This allows legacy systems to connect using existing public key algorithms, even though the structure of the X-509 certificate changed. These certificates can be gradually integrated over time. The TLS handshake process, used for connecting to web servers using modern cryptosystems and TLS certificates, involves negotiation to determine the encryption algorithm used. Post quantum cryptography follows this process, except when upgrading systems to quantum safe crypto algorithms and using hybrid certificates.

4. Engineering Standard

4.1. Environmental and Health/Safety Concerns

4.1.1. Environmental Concerns

While quantum computers may be more efficient simply on a processor function basis, they take a lot of auxiliary devices such as a cooling apparatus in their current iteration. Therefore the total energy consumption of a quantum computer and its functional apparatus would be significantly larger than a traditional computer. While this may be worse for the environment, our software does not have any impact on this. The impact of quantum computing's energy consumption is only a concern for those who are trying to brute force our authentication.

4.1.2. Health/Safety Concerns

Quantum Cryptography poses a threat to the healthcare industry due to its capability of breaking encryption used for sensitive data. A patient's data needs to be protected from potential outside attacks. This is to ensure the safety of all patients. Additionally, it introduces a safety concern to all encryption methods. In a general sense, people's public data is at risk as they innocently browse the internet. Other high end risks include banks and government related data. PQC could lead to a breach of security to endanger people's money and personal information. All forms of non post-quantum encryption are under threat of quantum computing developments.

4.2. Social, Political, and Ethical Concerns

4.2.1. Social Concerns

In the digital age, personal and sensitive information is increasingly stored and transmitted online. By safeguarding data against potential quantum-computing threats, the project ensures that personal information, including financial details, health records, and private communications, remains confidential. This is crucial for maintaining autonomy of individuals who want to be assured their private information and lives remain untouched by unauthorized entities. By implementing quantum-resistant cryptographic algorithms with our library, the project aims to ensure the confidentiality and integrity of personal data against quantum attacks, thus safeguarding the personal information and privacy of individuals in the digital space. Most importantly, consumers of technology should be confident in their data confidentiality and privacy.

4.2.2. Political Concerns

Currently, there exist various options for citizens who would like to have encryption in their data to protect their privacy. Adding post-quantum encryption will further this privacy for the users of our software. Politically this reduces groups and organizations from being able to access data and enhances the effort to data privacy. In many countries governmental laws require a software backdoor, this serves a significant risk to the success of this project. We have decided not to add any sort of backdoors to the system as this would weaken encryption and lower the overall privacy of users.

4.2.3. Ethical Concerns

The primary ethical concern which has been mentioned by our sponsor is open source licensing. All of the libraries that we have used thus far have an MIT license governing their usage. This is ideal because it allows our sponsor to use the software that we create without needing to be concerned about additional commercial licensing issues.

4.3. Manufacturability, Sustainability, and Economics

4.3.1. Manufacturability

When creating our product, we want to keep in mind the architecture and design that we choose to implement. We want to create a product that is easily replicable and adaptable to whatever needs the enterprise has. Ensuring that the project is clearly understandable and easy to replicate enhances the manufacturability. We are able to achieve this through the use of an open-source software. By using technology that is widely available, this allows us to increase the scalability potential of the project.

4.3.2. Sustainability

This project's focus on creating long-lasting solutions through PQC aligns with the idea of sustainability by aiming for solutions that remain effective over time without necessitating frequent, resource-intensive updates. By focusing on the durability of solutions, the project aims to create cryptographic systems that are not just reactive to current threats but are inherently resilient against future technological advancements. The emphasis on long-term efficacy is also a reflection of an efficient use of resources. Frequent updates not only demand significant computational resources but also require substantial human and financial investment. This strategy minimizes resource expenditure and future financial strain on organizations and individuals who rely on this technology.

4.3.3. Economics

The economics of this project, particularly through an efficiency lens, are influenced by the strategic decision to utilize open-source software. One of the most direct economic benefits of leveraging open-source software is the substantial cost savings compared to proprietary alternatives. Open-source software typically comes with no licensing fees, which significantly reduces upfront and ongoing costs for the project. This allows for a more efficient allocation of the budget, freeing up financial resources that can be redirected to more critical areas.

The open source model also inherently lowers barriers to entry in the digital market. By providing access to the software's source code, it enables a wider range of developers and enterprises to engage with and contribute to the project. Lower barriers to entry stimulate competition and innovation, which are key drivers of economic efficiency and growth.

5. Requirements

5.1 Overview

Our solution seeks to provide an importable Go library that requires minimal build and install criteria. This library should provide functionality for creating, parsing, and verifying PQ X.509 certificates for any application that wishes to use them for authentication. We also seek to provide valuable insight in how post-quantum cryptography performs relative to existing classical algorithms so that web admins can make informed decisions on what to use for security.

Users should be able to look at the data we provide and the library we provide to determine whether or not they wish to include post-quantum cryptography in their architecture. A user that chooses to import our library should simply be able to import our package and perhaps run a setup script to be able to use it.

5.2 User Stories

This project will require many different user stories to be completed in order to say we have accomplished the purpose of the project. Following, in **Table 5.2.A**, we have provided a short list of different stories and what they might look like. For an exhaustive list of our stories, visit **Annex 3**.

Table 5.2.A Example User Stories

Priority	User Story	Persona	Acceptance Criteria	Story Points
1	As a web admin, I want to be able to import quantum safe X.509 library into my package	web admin	I can start seeing functions after a simple "import" statement in my program	2
1	As a web admin, I want to be able to quickly set up the quantum safe X.509 library	web admin	I can start using the imported library after a quick build instruction or running a script that builds necessary library components	4
2	As a web admin, I want to be able to call quantum-safe X.509 library to generate a certificate	web admin	I can call a generate certificate function and get returned a valid certificate	4
12	As a client, I want to be able to verify a quantum-safe certificate	client	I can call a function that determines if a certificate is valid	2

13	As a client, I want access to documentation on how to build quantum-safe components	client	I can access manual pages for the quantum-safe library build	4

5.3 Requirements Models

For our system to meet all the set requirements, we need to develop a library that provides clients and servers with the functionality to securely authenticate and communicate with each other.

Figure 5.3.A below shows the necessary steps for a client and server to interact with one another. They both need to have a shared key in which they use to encrypt and decrypt messages from each other. This shared key needs to be agreed upon via a process known as Key Encapsulation (KEM). This is a mechanism that needs to take place after the client and server have authenticated each other and needs to be as secure as possible to ensure no foreign party also has access to the shared key.

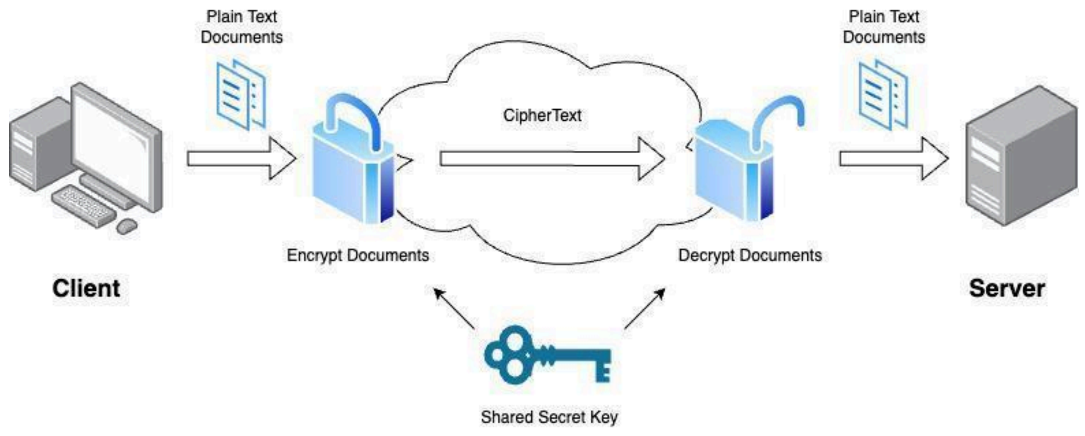


Figure 5.3.A Client Server Interaction

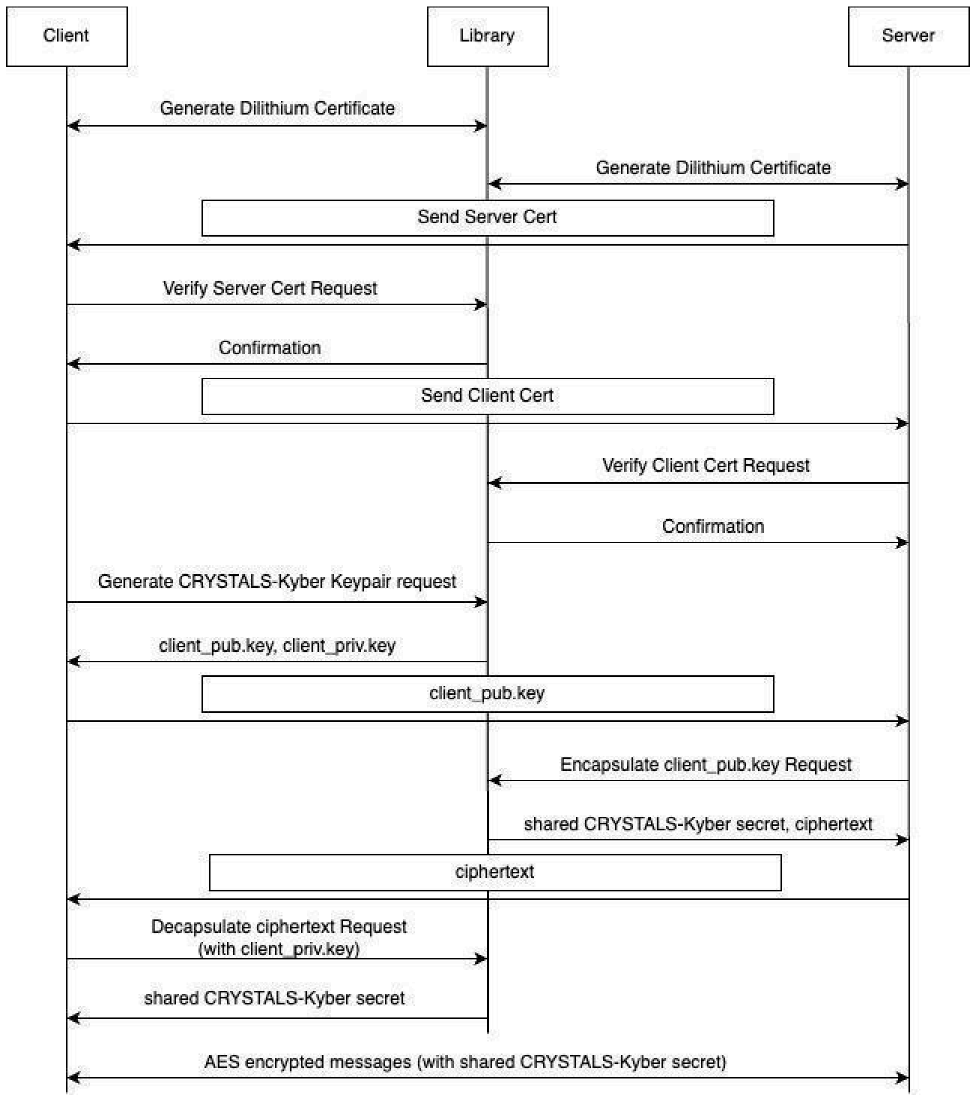

Client
Library
Server
Generate Dilithium Certificate
Generate Dilithium Certificate
Send Server Cert
Verify Server Cert Request
Confirmation
Send Client Cert
Verify Client Cert Request
Confirmation
Generate CRYSTALS-Kyber Keypair request
client_pub.key, client_priv.key
client_pub.key
Encapsulate client_pub.key Request
shared CRYSTALS-Kyber secret, ciphertext
ciphertext
Decapsulate ciphertext Request
(with client_priv.key)
shared CRYSTALS-Kyber secret
AES encrypted messages (with shared CRYSTALS-Kyber secret)

Figure 5.3.B Authentication Flow Diagram

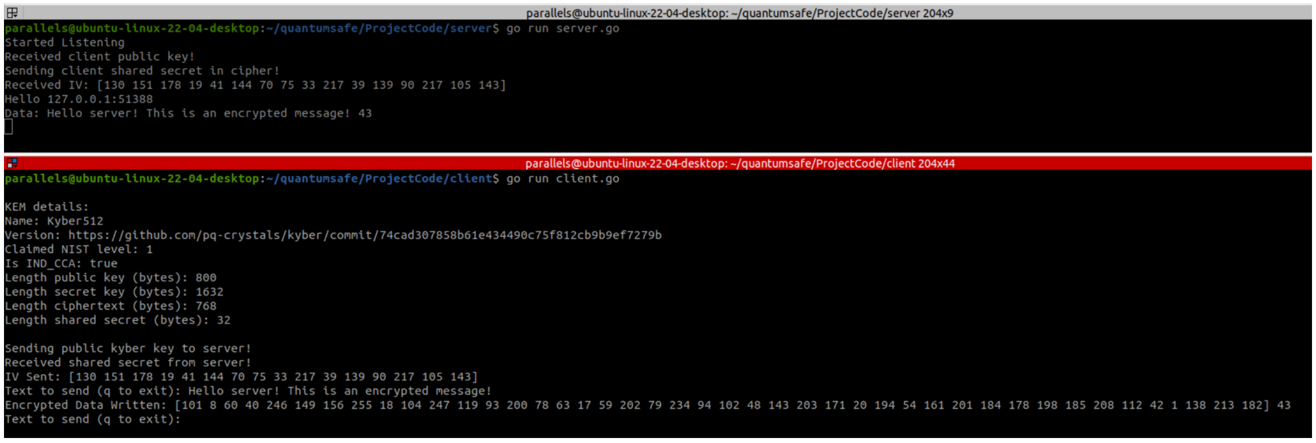
Figure 5.3.B above shows how we will need to be able to securely get this shared key on both the client and server. There are three main parts of this sequence, namely: Authentication, KEM, and AES. During authentication, the client and the server generate and sign a dilithium certificate. This certificate proves that the client and server are who they say they are. They exchange certificates and verify them using functions provided in our library. At this point, the client knows for a fact that they are communicating with the server and vice versa.

Next, during KEM, the client and the server discuss a shared secret. They agree on an algorithm to use, such as the post-quantum Kyber512 algorithm, and then generate a key only they know which is used during the final step, AES.

In AES, the client and the server encrypt and decrypt messages with the shared secret key developed during KEM. This is what keeps the communication completely confidential.

5.4 Prototypes

Much of our design will rely on terminal based UI for the client/server tunnel application. Clients should be able to dial into a server and begin exchanging encrypted messages after the authentication process, prompts will look like the following figure:



```
parallels@ubuntu-linux-22-04-desktop: ~/quantumsafe/ProjectCode/server 204x9
parallels@ubuntu-linux-22-04-desktop:~/quantumsafe/ProjectCode/server$ go run server.go
Started Listening
Received client public key!
Sending client shared secret in cipher!
Received IV: [130 151 178 19 41 144 70 75 33 217 39 139 90 217 105 143]
Hello 127.0.0.1:51388
Data: Hello server! This is an encrypted message! 43
```

```
parallels@ubuntu-linux-22-04-desktop: ~/quantumsafe/ProjectCode/client 204x44
parallels@ubuntu-linux-22-04-desktop:~/quantumsafe/ProjectCode/client$ go run client.go

KEM details:
Name: Kyber512
Version: https://github.com/pq-crystals/kyber/commit/74cad307858b61e434490c75f812cb9b9ef7279b
Claimed NIST level: 1
Is IND_CCA: true
Length public key (bytes): 800
Length secret key (bytes): 1632
Length ciphertext (bytes): 768
Length shared secret (bytes): 32

Sending public kyber key to server!
Received shared secret from server!
IV Sent: [130 151 178 19 41 144 70 75 33 217 39 139 90 217 105 143]
Text to send (q to exit): Hello server! This is an encrypted message!
Encrypted Data Written: [101 8 60 40 246 149 156 255 18 104 247 119 93 200 78 63 17 59 202 79 234 94 102 48 143 203 171 20 194 54 161 201 184 178 198 185 208 112 42 1 138 213 182] 43
Text to send (q to exit):
```

Figure 5.4.A Basic Client Tunnel UI

The library we are providing will be filled with many different Go functions that generate relevant, quantum-safe cryptography components. These functions will be mostly static calls that perform some sort of required functionality for the client or server and can be called in a manner represented in **Figure 5.4.B** below:

```go
package main

import(
    "github.com/CSCE482QuantumCryptography/qs509"
)

func main() {
    cert, err := qs509.GenerateCertificate("dilithium5", "cert.out")

    conn, err := net.Dial("tcp", "127.0.0.1:8080")

    conn.Write(cert)

}
```

Figure 5.4.B Basic Client Generate Certificate

6. Design

6.1 Overview

Our solution will be using a number of different tools and technologies in order to create a fully functional library and testing suite. Starting off with the library, we will be leveraging three different post-quantum solutions that work together to provide us with the necessary resources to export our own Go wrapper:

- OpenSSL 3.3 (post-quantum release)
- LibOQS
- oqs-provider

Together, these three softwares allow us to generate and export X.509 certificates, verify them, and parse through their information. They implement a variety of post-quantum algorithms to sign and encrypt data with, our team will focus mainly on Dilithium3 and Kyber512, though we have added functionality for more. As for our testing suite, we will require a few more technologies:

- Mininet Virtual Machine
- Wireshark

These two work in conjunction to set up and measure performance for a variety of network topologies and show exactly how our solution looks as packets move across the network. Our team uses personal computers for everything outlined above, and further design requirements are explored in **Annex 1**.

6.2 Comparison of Potential Solutions

In our project, we were given discretion on how we want to implement our post-quantum library. We were given the choice on what algorithms to offer for signing and encrypting, with the idea that as we progress we start to offer a variety of choices.

OpenSSL 3.3 and LibOQS offer a large variety of signing algorithms, from CRYSTALS-Dilithium to ML-DSA, Falcon, etc. We focused on the Dilithium signatures to start off our library. This algorithm is a NIST approved front runner in the world of post-quantum signing algorithms and is why we are choosing to pursue this algorithm for our library's basic functionality when signing and verifying certificates.

As for our encryption, we used the provided CRYSTALS-Kyber KEM suite. This follows the same reasoning, Kyber is one of the most well known quantum-safe encryption algorithms and produces shared keys that are compatible with AES256 encryption.

There are many different virtual machine systems and frameworks we can use to simulate network topologies, some provided by Apposite, Network Delay Simulator, Network Link Configurator (MacOS), etc. However, we have chosen to pursue Mininet for our simulation. This free software provides an image for virtual machine hosts that comes pre configured with everything needed to simulate and measure different networks. It is very quick and easy to set up and provides documentation for setting up networks.

6.3 Data Design

Throughout our system we will be interacting with a few different data points. Most importantly, we will be working with certificates and keys throughout the authentication process. These can be related to one another as shown in **Figure 6.3.A** below:

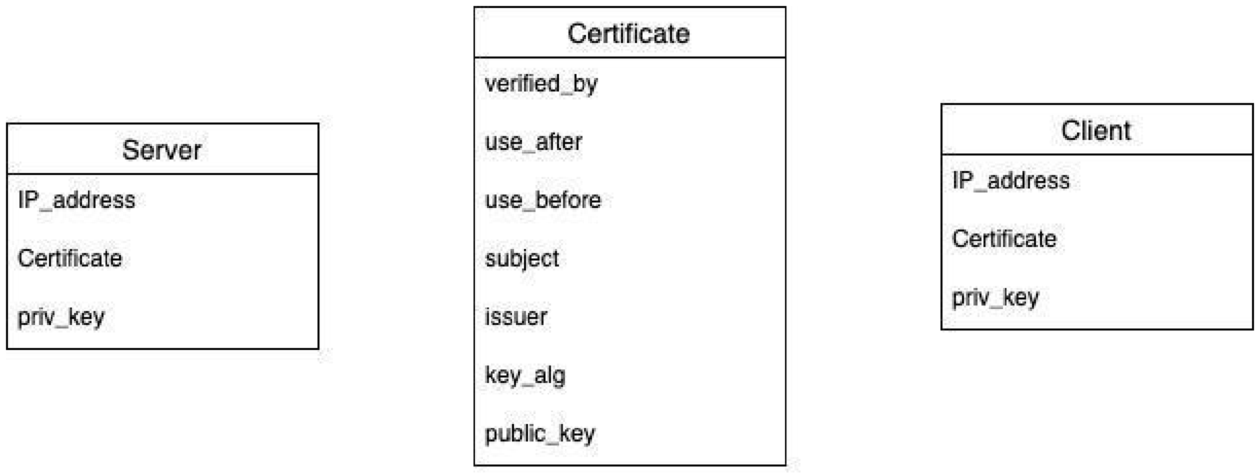


Figure 6.3.A ERD Diagram

Keys can be different sizes, depending on what algorithm was used to construct them and whether they are private or public. However, they will always be stored as some sort of []byte variable. Certificates are their own object and contain a ton of different information that we will need to parse through, such as the issuer, the expiration date, etc.

Clients and servers are set up very similarly. They each will have their own certificates and keys that were generated by our qs509 library. They will then be able to parse through the certificate attributes in order to verify incoming certificates and their authenticity.

6.4 Functional: Structural Design

Figure 6.4.A (shown below) outlines the structure we intend to use for our Go qs509 library. We have split up the library into 3 different major sections, namely: generating certificates, verifying certificates, and benchmarking the performance.

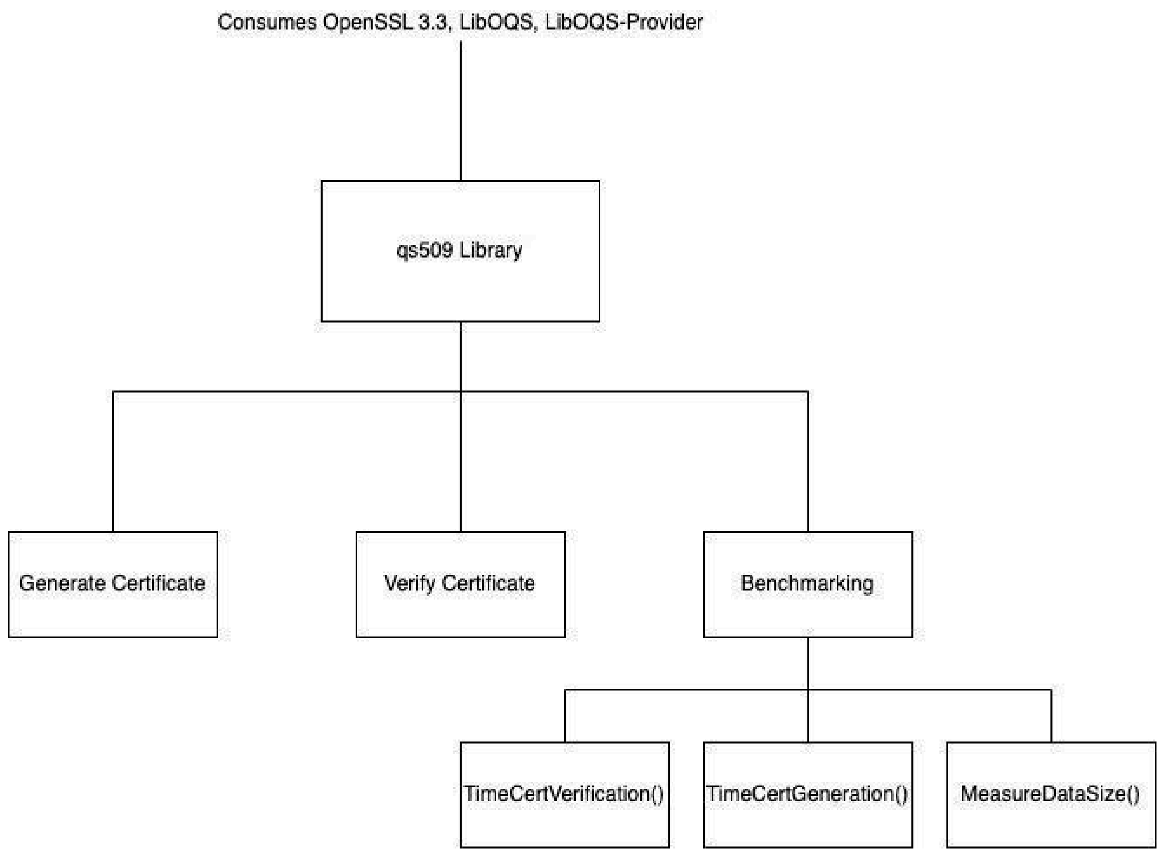


Figure 6.4.A qs509 Library Structural Diagram

This library can be consumed by any other Go package and used to call the functions above.

Along with the qs509 library, we will be providing server and client tunneling code that implements our library to demonstrate how it can be used. Figures **6.4.B** and **6.4.C** outline the structure of the tunneling.

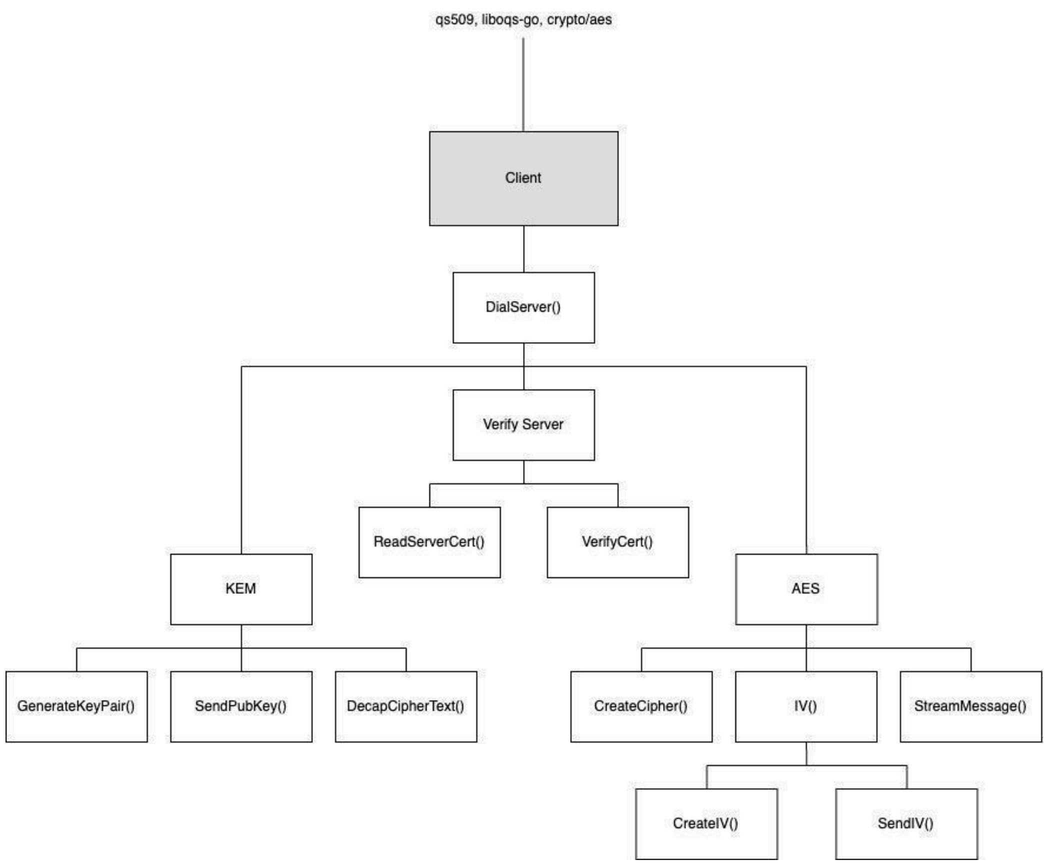


Figure 6.4.B Client Structural Diagram

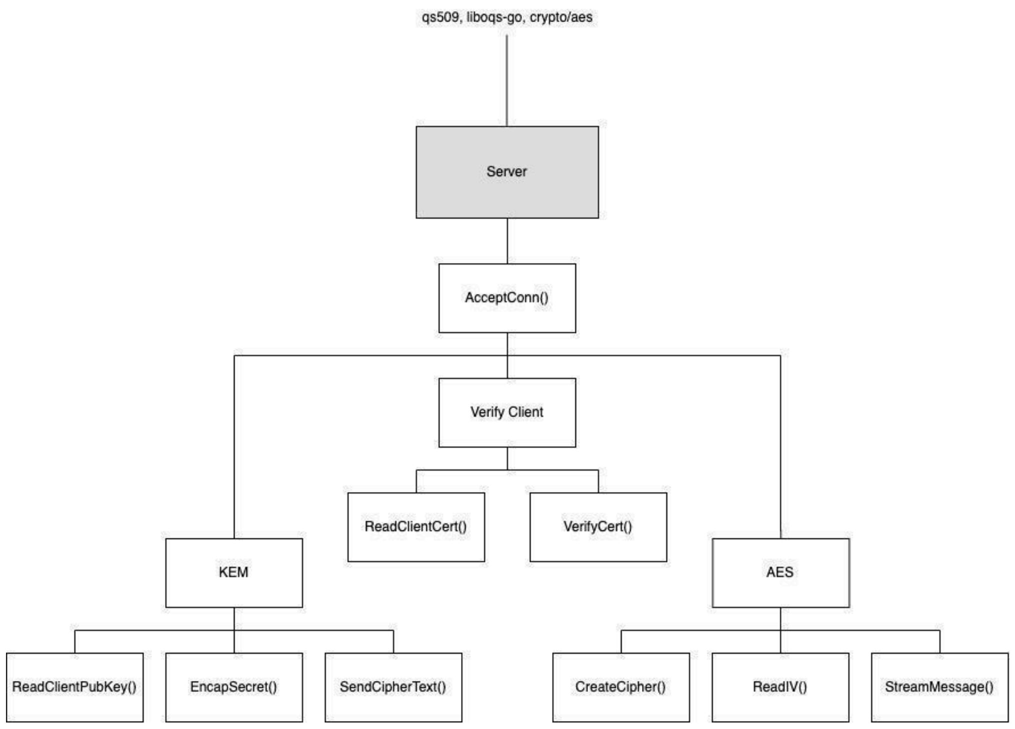


Figure 6.4.C Server Structural Diagram

6.4.1 External Interfaces

Our tunneling application will be utilizing a few existing APIs for functionality with KEM and AES. OQS provides a go wrapper for post-quantum KEM, including the following API calls:

- ReadClientPubKey()
- EncapSecret()

These calls take in information about which post-quantum algorithm to use and then return the information requested.

The standard go library, “crypto/aes” provides calls for:

- CreateCipher()
- StreamMessage()

These calls return objects that are then used to stream messages across the network using the shared secret key that is generated during the KEM process.

The library we are creating will contain API calls for:

- GenerateCert()
- VerifyCert()
- ParseCert()

These calls will be used to generate certificates, take in existing certificates and parse through them, and take in certificates and verify if they have been signed by a provided certificate authority.

6.5 Functional: Dynamic Design

The most important part of our tunneling system is the authentication and authorization process of a client connecting to a server. The following sequence diagram in **Figure 6.5.A** outlines this process:

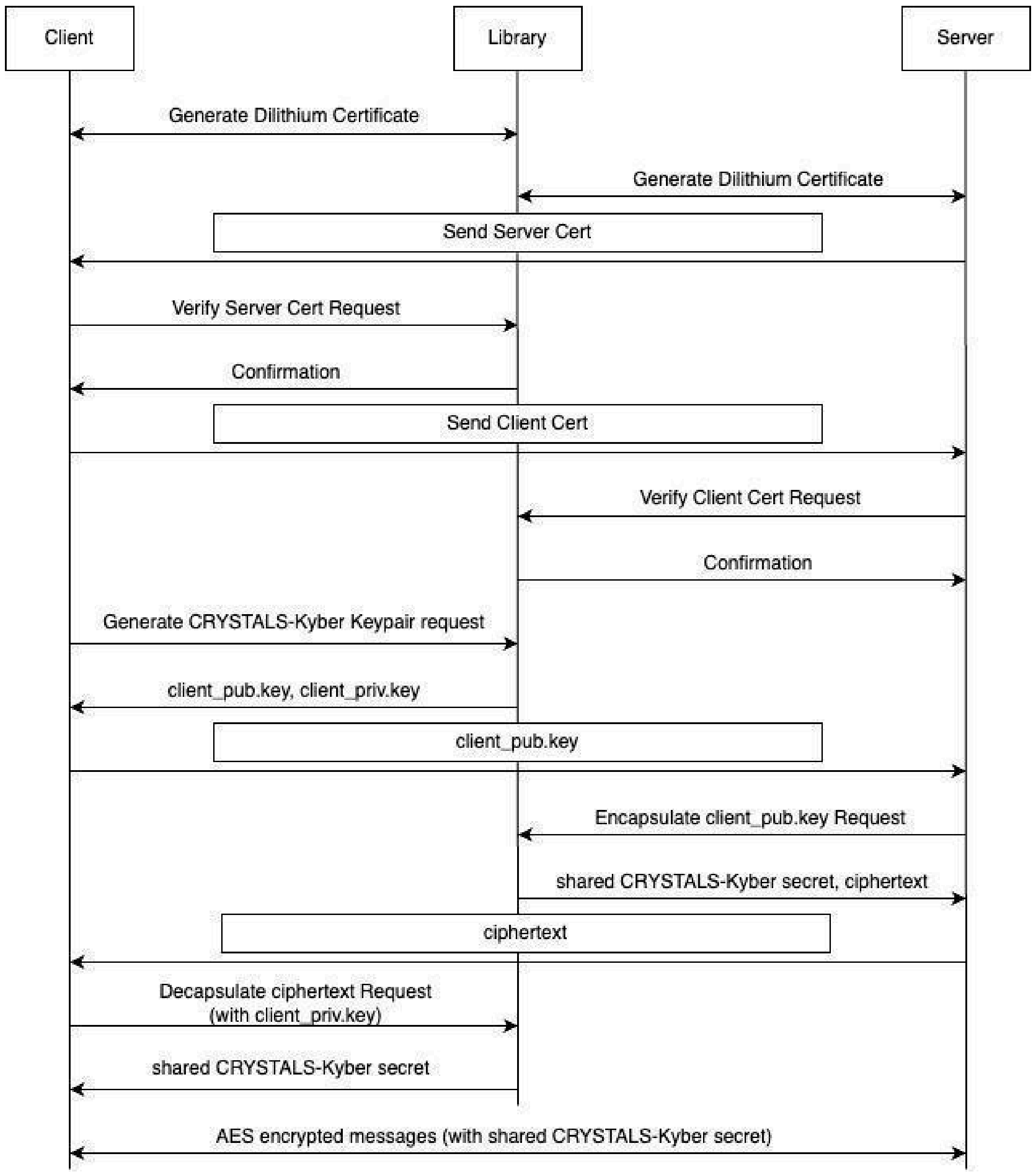

Client
Library
Server
Generate Dilithium Certificate
Generate Dilithium Certificate
Send Server Cert
Verify Server Cert Request
Confirmation
Send Client Cert
Verify Client Cert Request
Confirmation
Generate CRYSTALS-Kyber Keypair request
client_pub.key, client_priv.key
client_pub.key
Encapsulate client_pub.key Request
shared CRYSTALS-Kyber secret, ciphertext
ciphertext
Decapsulate ciphertext Request (with client_priv.key)
shared CRYSTALS-Kyber secret
AES encrypted messages (with shared CRYSTALS-Kyber secret)

Figure 6.5.A Authentication Sequence Diagram

7. Evaluation

7.1 Overall Evaluation Plan

For our project, we have defined a definition of success, which is the criteria required to say we have completed our project. This definition is:

- At a minimum, our job is not complete without the completion of a Go wrapper for PQ X.509 certificate generation, verification, and parsing that is used to demonstrate a successful client/server encrypted tunnel across different network topologies.

To test this definition, we have a variety of internal and external tests laid out with criteria used for examining them, these are explored in detail later in sections **7.2** and **7.3**.

In short, we plan to use unit and integration tests to say from an internal perspective we have accomplished this goal. These tests will be designed by us to cover as much code as possible and display metrics (such as overall code coverage %) that prove we have done so.

Once we have completed internal testing, we will move on to external testing by way of User Acceptance Testing. Our user acceptance testing will revolve around usability as reported by our sponsor. Any additional users would be gathered from the sponsor company. Because this software is not designed to be client-facing, we do not expect intuitive usability without documented steps laying out the process behind using our system.

Once the users have had time to use our solution, we would conduct surveys to acquire information regarding their acceptance of the solutions, areas that could be improved, areas that work great, and determine if overall they agree we have addressed the problem our project aims to solve.

7.1.1 Quality Plan

We have broken down our system into a few different quality goals that must be achieved in order to say we have completed our project. These quality goals are:

- Create a system that can easily be interacted with and has favorable feedback (measured in user surveys) from at least 95% of users by the end of April
- Create an architecture that allows clients to communicate with a server via 100% quantum encrypted, authenticated communication by the end of April
- Capture meaningful performance metrics during every portion of the cryptography, tunneling, and authentication to give us a benchmark on how our system compares to other classical implementations by the end of April

These quality goals are rather broad and encapsulate different portions of the code, so to ensure that we are meeting these goals we have developed Quality Policies that everyone on our team must follow when coding. These policies are:

- Only use existing, expert-approved post-quantum implementations when dealing with cryptography
- Any code changes need to be reviewed and approved by two other group members before merging into the code
- Use ready made network simulators for performance testing over different network topologies

In following all of these policies, our group will use the following processes, or steps, that ensure we are at all times following the policies we have set to meet our goals:

- Code changes should occur on a development branch on GitHub. Anytime there are changes we would like to move over, we need to make a PR request.
- PR requests should be set to require at least 2 additional reviews and approvals in order to be merged into main.
- All unit tests should cover at least 100% of code.
- Any UI/UX changes should be reviewed with the team to ensure intuitiveness
- Any code changes should be documented, with reasoning behind why the change was necessary

All of these processes will produce tangible, trackable artifacts that can be used to trace our progress towards completing our quality goals. These artifacts are:

- GitHub logs for PRs, approvals, comments
- Documentation for code changes and reasoning
- UI/UX approvals from teammates
- Test Cases in the source code
- Test Cases passing rate after being run in Ziti

7.2 Internal Evaluation Plan

We will use unit testing to test the specific elements of our system as they work individually, rather than as a whole. Sections such as key generation will be tested by themselves to prove they are working correctly outside of the tunneling network by using functional unit tests. Of course, unit testing will aim to cover all the code we have implemented within our Go Library for quantum-safe X.509. Some example unit test cases follow:

Test Case	Input	Expected Output
Sunny Test: LibOQS cert generation	User specifies a valid key to generate a certificate	Quantum Safe certificate exported to user
Rainy Test: LibOQS cert generation	User specifies an invalid key to generate a certificate	Program terminates after saying the key is invalid
Sunny Test: Verify certificate	User gives a valid certificate to OpenZiti to Verify	Program authenticates the user to the router
Rainy Test: Verify certificate	User gives an invalid certificate to OpenZiti to Verify	OpenZiti rejects the certificate with an error code, user is not authenticated

Table 7.2.A Unit Test Examples

Once the individual parts are tested via unit testing, we will move on to integration testing, making sure that the different modules can come together to form one fully functional program. This testing will explore the flow of data from one component to another, authentication to the network, etc.

The biggest part we need to focus on for integration testing is the connection between our post-quantum implementation module (LibOQS) and our tunneling application. We will need to test that once we have generated a token or a key in LibOQS, that item is not changed as it moves throughout the tunnel. Some integration test cases follow:

Test Case	Input	Expected Output
Sunny Test: Generate certificate and give to server	User specifies a valid key to generate certificate and presents it to the server	A certificate is exported to the user and then verified by the server
Rainy Test: Generate certificate and give to server	User specifies an invalid key to generate a certificate and presents the certificate to the server	The certificate is invalid and the user is not authenticated

Table 7.2.B Integration Test Examples

A more comprehensive list of both unit and integration tests is available in Annex 7.

7.3 User Acceptance Test Plan

In the end, our project will not be complete without the approval of those who will be using it as consumers. Therefore, we want to make sure that we have a thorough user acceptance test plan to gather information from users and prove that we have a working solution that meets their standards.

To conduct our user acceptance testing, we will be constantly demonstrating our project to our sponsors, L3Harris, as we develop. This will enhance the result of our project and help them understand the next phase in our user acceptance testing, deploying our project themselves.

Our team will create documentation that explains how to compile and launch our finished network and library. This documentation will get turned over to our sponsors and they will get some time to try the solution themselves. They will be instructed to document any frustrations, errors encountered, anything that pleases them, etc.

Finally, we will interview the users and get honest feedback about our system and what it accomplishes.

7.3.1 Recruitment of Users

Due to the technical nature of our project, the users for our testing will be from the sponsor company, L3Harris. Our goal is to have the group of sponsors we have been working closely with evaluate our system and its completeness. The users are expected to:

- Read the documentation for build instructions
- Download and install our tunneling source code on their machine
- Launch the network by creating a client and server
- Authenticate the client with the server to begin sending encrypted messages

All along the way, the users should be documenting any frustrations they run into, errors they encounter, and anything overall they would like to talk about. Afterward, an interview with the development team will be used to discuss the results of the testing.

7.3.2 User Acceptance Test Artifacts

The test artifacts to be produced by user acceptance testing include documentation from users about their experience working with the system, meeting minutes during our structured interview with the users after they have used the system, and survey results that gauge how intuitive our system was to use and whether or not the users agree it solves the problem it is meant to solve.

7.3.2.1 Protocol

During user testing, participants will be instructed to follow the documentation we provide as best as possible. They should move through the instructions chronologically to compile and deploy their own post-quantum network. Once this happens, they should be able to confirm the network is running and that it does what it is supposed to.

Afterward, participants should respond to our survey with honesty and elaborate on any of the open-ended questions asked. They should think of this survey as a way to communicate back to us anything we can improve for our system.

7.3.2.2 Survey

For our testing, we will rely on surveying to get an understanding of how the users feel about our system. The survey will be hosted through Qualtrics and will be used to evaluate all the different parts of our system.

Some questions will be structured in a way that allows users to rate different aspects of the solution on a scale of 1 to 5 (1 is strongly disagree, 5 is strongly agree), example prompts are shown below:

- The UI/UX is easy to navigate
- The network is easy to build
- Post-quantum keys are used in the correct capacity

Other questions may be more open-ended, allowing the users to elaborate on specific features, the features they like or dislike, anything that is not intuitive enough, etc. Examples listed below:

- What parts of setting up the network seemed intuitive?
- What parts– if any– of network setup seemed confusing?
- Does the tunnel program follow your understanding of how networks should be set up?

A more exhaustive list of questions can be found in Annex 8.

8. Implementation

Our solution has three different major components, each of which will require its own implementation strategy and will leverage different technologies. We will dive into each component separately, these components are:

- Go library for OQS X.509
- Client/Server Tunneling Application
- Mininet Simulation

Of course, managing and integrating these different components will be its own effort, this is more explored in Annexes 1 and 2

8.1 Go Library for OQS X.509

Open Quantum-Safe is a framework that offers functionality for post-quantum X.509 and key exchange. However, this framework is only available for use within some sort of UNIX terminal, and not for use in language specific SDKs like Go. Therefore, this Go library takes the necessary commands from OQS and provides Go functions that wrap around them so they can be used by Go applications.

OQS requires two other packages itself to build and run. These packages are:

- OpenSSL 3.3
- oqs-provider

Therefore, to implement our library, we needed to follow setup instructions for these packages. Luckily, IBM offers a quick setup guide for installing these packages.

With everything installed, we began setting up our own package to be exported. This required us to create a package for qs509. We initiated the package by running the command:

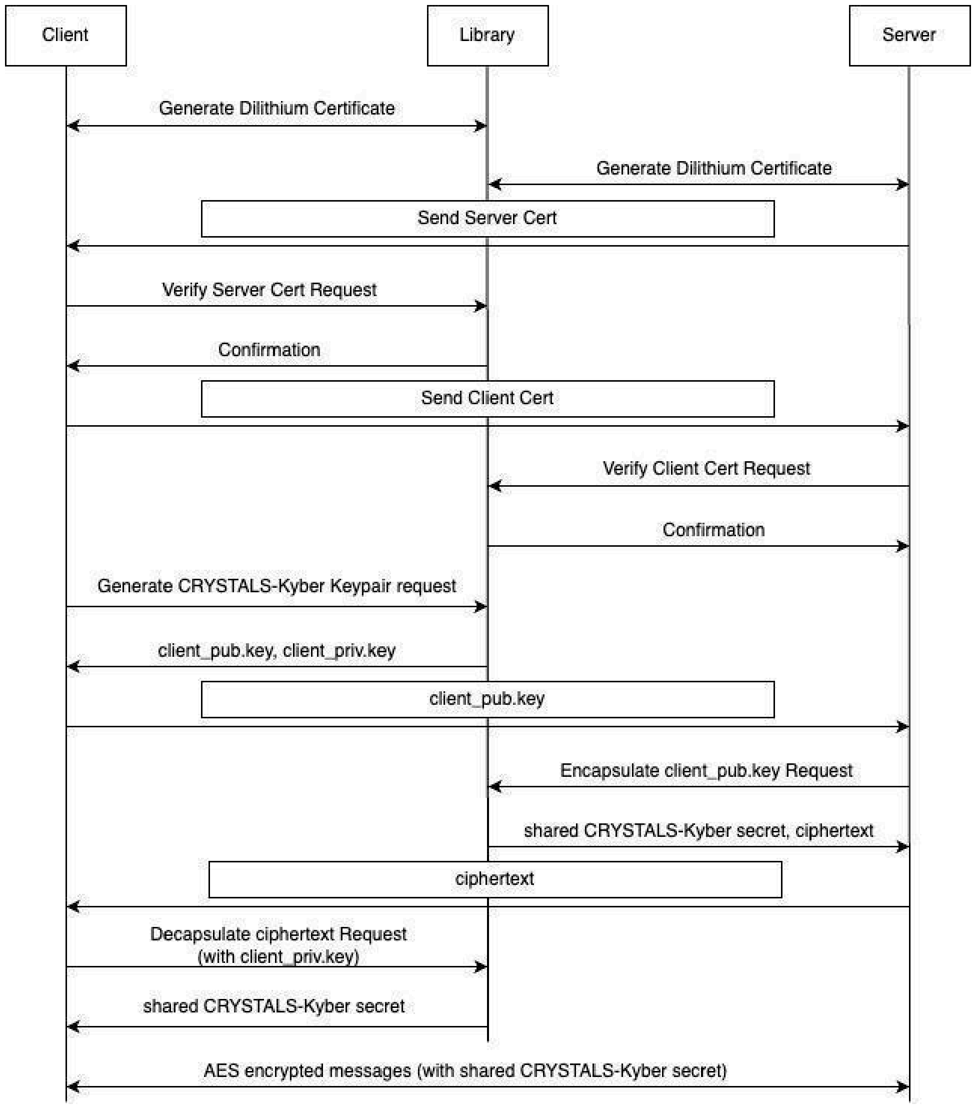
```
go mod init "github.com/CSCE482QuantumCryptography/qs509"
```

With our mod file created, any other Go application should be able to import our package by importing the name of our module. While this is great, it doesn't mean anything until we develop functionality for our library.

We added Go functions for generating certificates, verifying them, parsing them, etc. This required us to use the standard go "os" import and execute commands out to OQS. This then allowed us to create functions that can be called the same way any other Go function is called and save the consuming applications from having to execute the commands in a non-friendly way.

8.2 Client/Server Tunneling Application

The client/server tunneling application we provided is a way to test our library and show that it can be used to ease the implementation of post-quantum X.509 verification. This library follows the following sequence:



Client
Library
Server
Generate Dilithium Certificate
Generate Dilithium Certificate
Send Server Cert
Verify Server Cert Request
Confirmation
Send Client Cert
Verify Client Cert Request
Confirmation
Generate CRYSTALS-Kyber Keypair request
client_pub.key, client_priv.key
client_pub.key
Encapsulate client_pub.key Request
shared CRYSTALS-Kyber secret, ciphertext
ciphertext
Decapsulate ciphertext Request (with client_priv.key)
shared CRYSTALS-Kyber secret
AES encrypted messages (with shared CRYSTALS-Kyber secret)

Figure 8.2.A Sequence Diagram

There are three different parts of the client/server tunnel:

- PQ X.509 Authentication
- Key Exchange
- AES256 Encryption

8.2.1 PQ X.509 Authentication

During this phase, the tunneling application makes good use of our qs509 library implemented in **8.1**. This library enables the client to call a function “GenerateCertificate()” and specify which signing algorithms to use on the certificate. The same is true for the server.

The client and server then exchange these generated certificates and once again use our library to verify their authenticity with a “VerifyCertificate()” function. This step ensures both the client and the server that they are talking to who they think they are talking to, and works to protect against man in the middle attacks.

8.2.2 Key Exchange

Once the client and the server know they are talking to the right entity, they need to agree on a shared secret to encrypt messages with each other. During this portion of implementation, both the client and server import the oqs-go KEM package. This library offers PQ KEM functions for creating, encapsulating, and decapsulating quantum-safe keys.

Firstly, a client calls “oqs.GenerateKeyPair()” to create a public and private key with a specified KEM algorithm supported by OQS. The client then uses the connection to the server to write over the generated public key in plain text.

Now, the server can read in the public key and use the “oqs.EncapSecret()” function to get a 32-byte shared secret that can be used for AES256 encryption. The server writes back this secret as a cipher to the client by encrypting it with the client’s public key.

Finally, the client can decipher this secret key by using its own private key and the “oqs.DecapSecret()” function. At this point, both the client and server have a shared, post-quantum secret that can be used in AES256 encryption.

8.2.3 AES256 Encryption

With the shared secret key, the client and the server can independently set up streams to write encrypted messages to one another and decrypt messages they receive. To do this, we made use of the “crypto/aes” library import which offers functionality for creating ciphers, randomizing them, and decrypting them.

The client and server each use “aes.NewCipher()” and the shared secret to create the baseline cipher. The client creates an initialized vector, which is used to ensure the cipher is completely random each time and works to protect it against anyone who may be listening in and trying to decode it. The client writes this IV over and the server accepts it.

At this point, any messages can be sent back and forth after being run through a “cipher.NewCFBEncrypter”. The messages are completely encrypted and the only way to decrypt them is to own the secret shared key that only exists on the client and the server.

8.3 Mininet Simulation

The Mininet simulation works to provide us with a variety of network topologies so we can test the above client/server programs on different networks. Implementing the simulation starts with the building and configuration of Mininet. This free software provides images of virtual machines that come pre-loaded with Mininet. Simply following the setup guide on Mininet will allow us to start our simulation.

We need to define our own topologies to actually run our different test cases and measure the time it takes for them to complete. To do this, we will be defining various topologies using the Mininet Python SDK. This is demonstrated in **Figure 8.3.A** below:

```python
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def build( self ):
        "Create custom topo."

        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's3' )
        rightSwitch = self.addSwitch( 's4' )

        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHost )


topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Figure 8.3.A Simple Mininet Topology

Once we have a topology defined, we can configure the different hosts to act as either clients or the service.

9. Results

The outcome of our project has 3 components. The first component is the qs509 library which we are producing. This library contains all of the necessary functionality to create, sign, and verify post-quantum X.509 certificates. Additionally, this library will leverage LibOQS in order to generate Kyber key pairs such that clients and servers can use them to generate secret shared keys. The second component is the client and server test programs. The client program will be able to connect to a server and both of the parties can leverage our library to create and verify one another's self-signed certificates. After verifying the identities of each entity, the clients and server can use the library to generate a shared secret key that is used to transmit messages with AES256 quantum-safe encryption. The last component of our project is performance evaluation and testing.

In the end, we found that there is a decent trade-off between performance and security when implementing these post-quantum algorithms in different network scenarios. First off, the sheer size of post-quantum cryptography components is vastly larger than classical counterparts. **Figure 9.A** illustrates this difference.

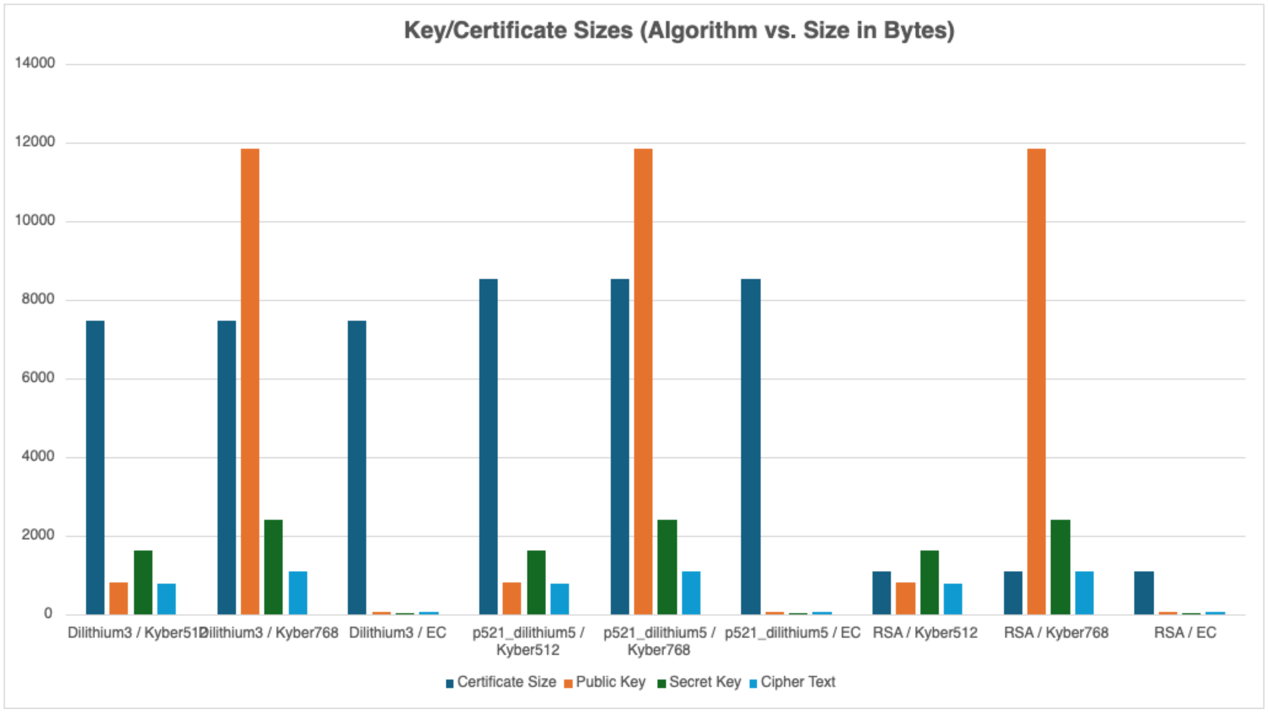


Figure 9.A Size difference in Bytes for different Components

We can see on the right side of **Figure 9.A**, the purely classical signing and KEM implementation has much smaller components than the purely or even the mixed implementations. With this in mind, we had predicted that the classical algorithms would almost always outperform the post-quantum implementations, but this isn't always the case.

In ideal network scenarios, the cost of having the larger certificates and keys is not felt that much, and post-quantum algorithms can perform on par with classical implementations for a variety of the different parts of our program.

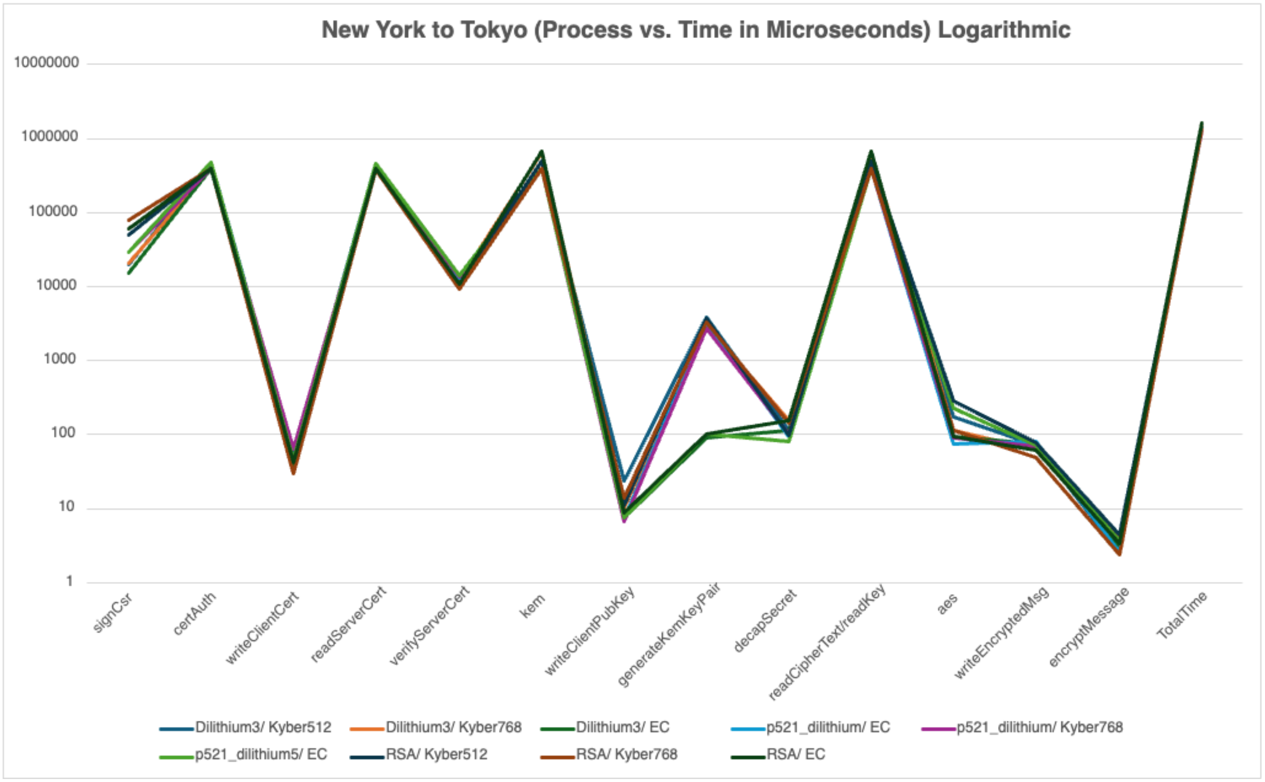


Figure 9.B New York to Tokyo Performance Results

For example, in **Figure 9.B**, we can see the time it takes (in microseconds) many different configurations of algorithms to perform different portions of the client/server code. We can see that the 9 different pairs of algorithms are for the most part very bunched together, completing tasks at a similar rate. Of course, the classical algorithms tend to generate keys a bit faster, but overall the programs terminate at around the same time.

However, when network conditions become less ideal, we can see that the classical algorithms tend to outperform the post-quantum ones more drastically.

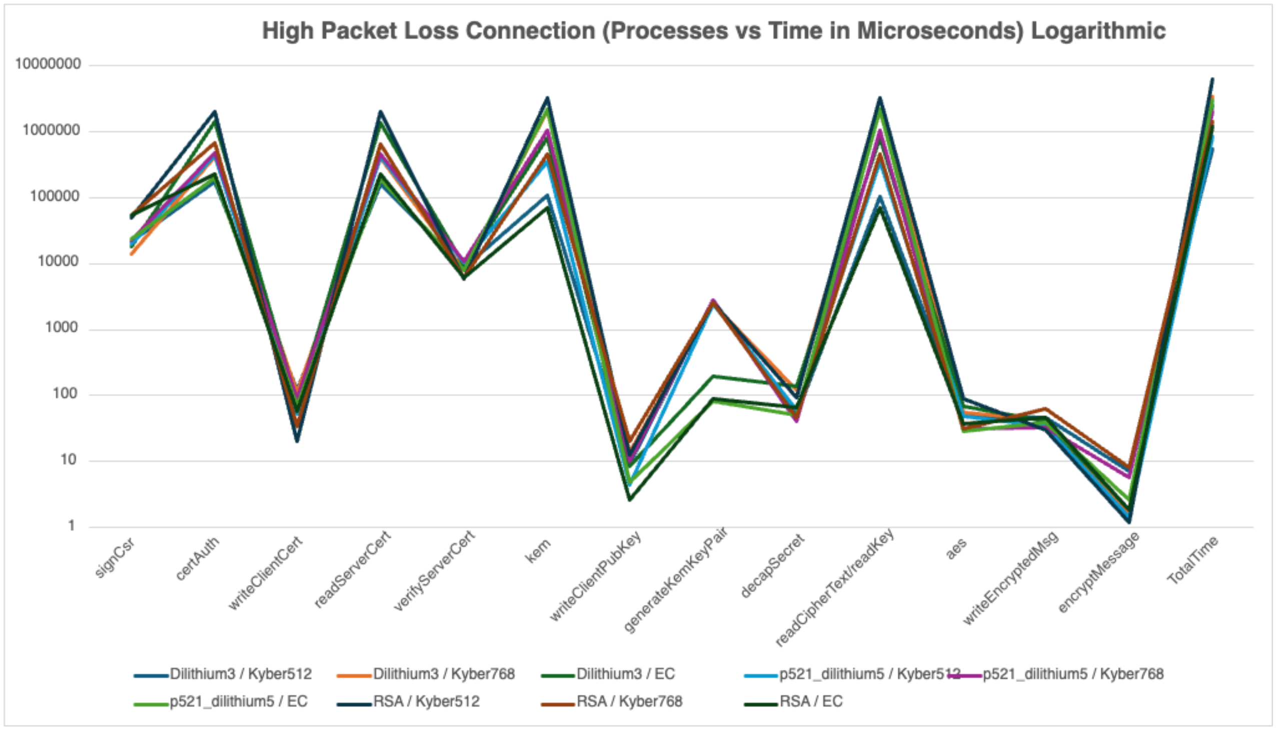


Figure 9.C High Packet Loss Performance Results

Figure 9.C highlights the devastating effect of having high packet loss on a network. This graph shows there is a much larger difference between the different pairs of algorithms, wherein the time it takes to send more packets across the network has a much larger effect on the performance because many of these packets get lost and then have to retransmit. Therefore, when there are more packets to transmit (like for post-quantum certificates), the time takes exponentially longer.

With this in mind, we take a closer look at the trade-off between size and time. **Figure 9.D** and **9.E** below explore this a little deeper. We can see that the time it takes to generate classical keys is faster across all the different network configurations tested. With this, they are also generating smaller keys of which can be transmitted across the network in fewer packets, allowing for faster communication. We can see however, in more ideal network configurations (such as New York to Los Angeles or low packet loss), the difference in write times between the post-quantum and classical algorithms is not as severe as in the less ideal cases.

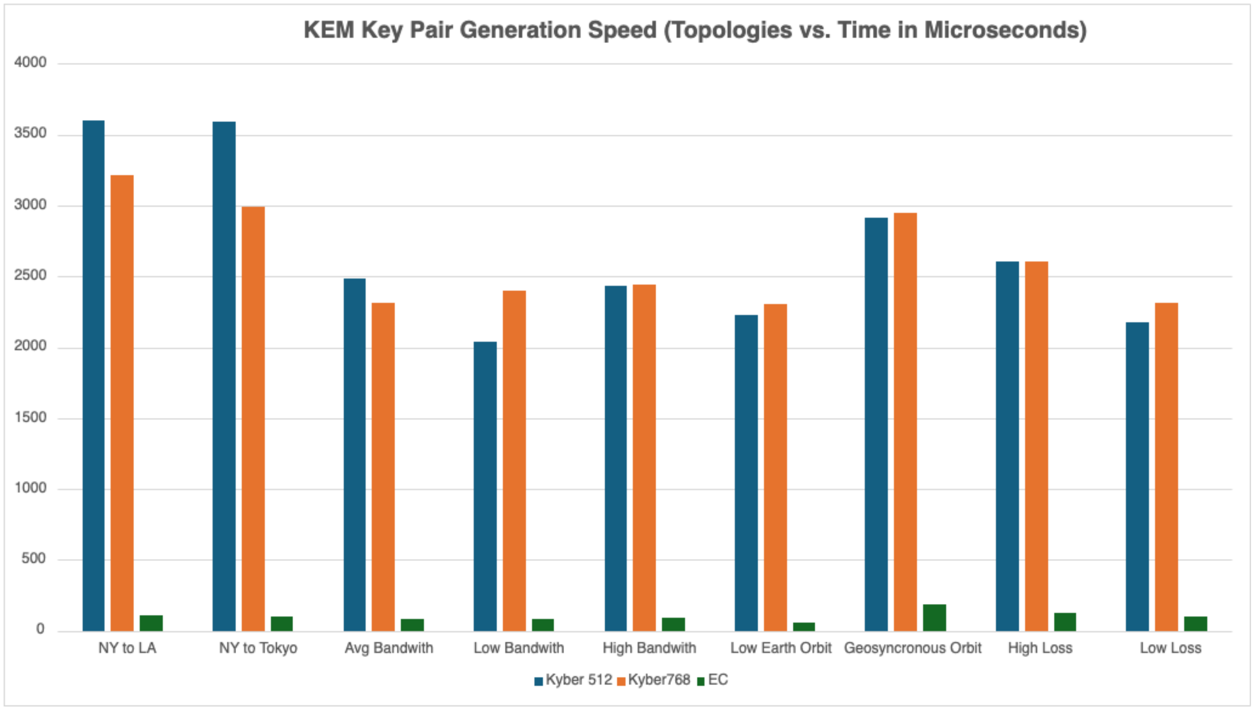


Figure 9.D Public Key Generation Time (Microseconds)

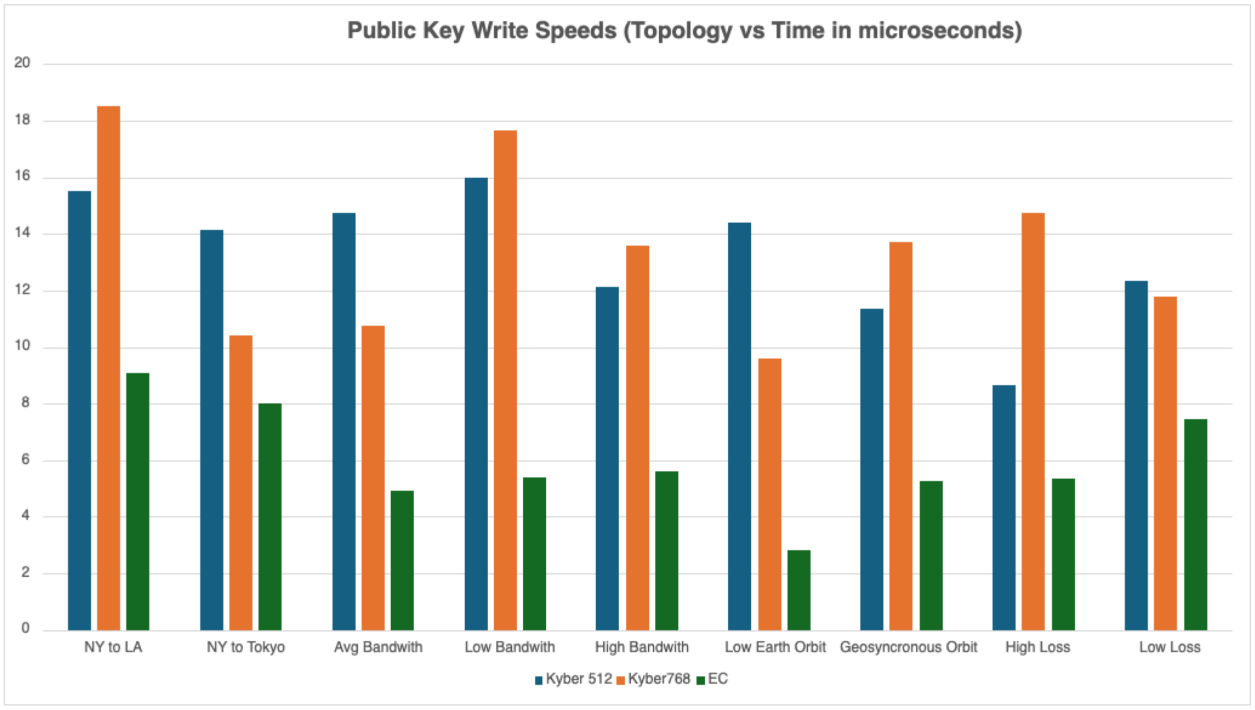


Figure 9.E Public Key Communication Time (Microseconds)

Additionally, we can look specifically at the certificate generation and write speeds.

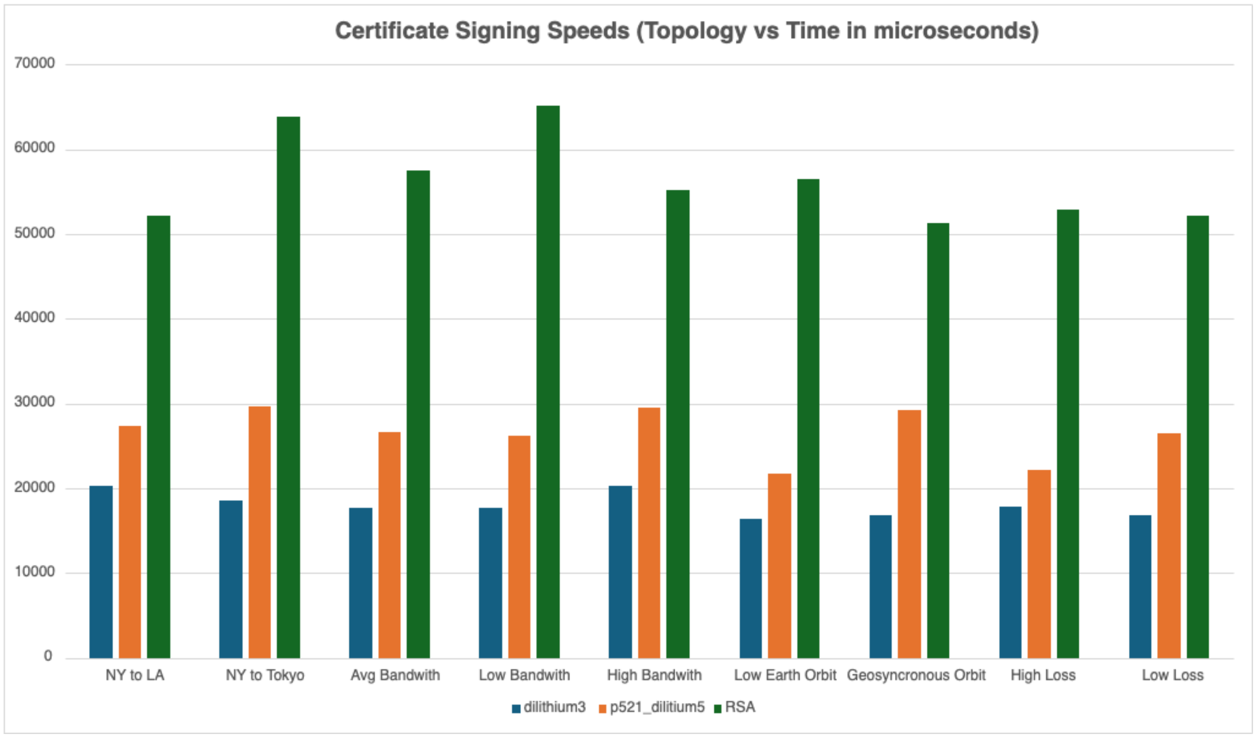


Figure 9.F Certificate Generation Speed (Microseconds)

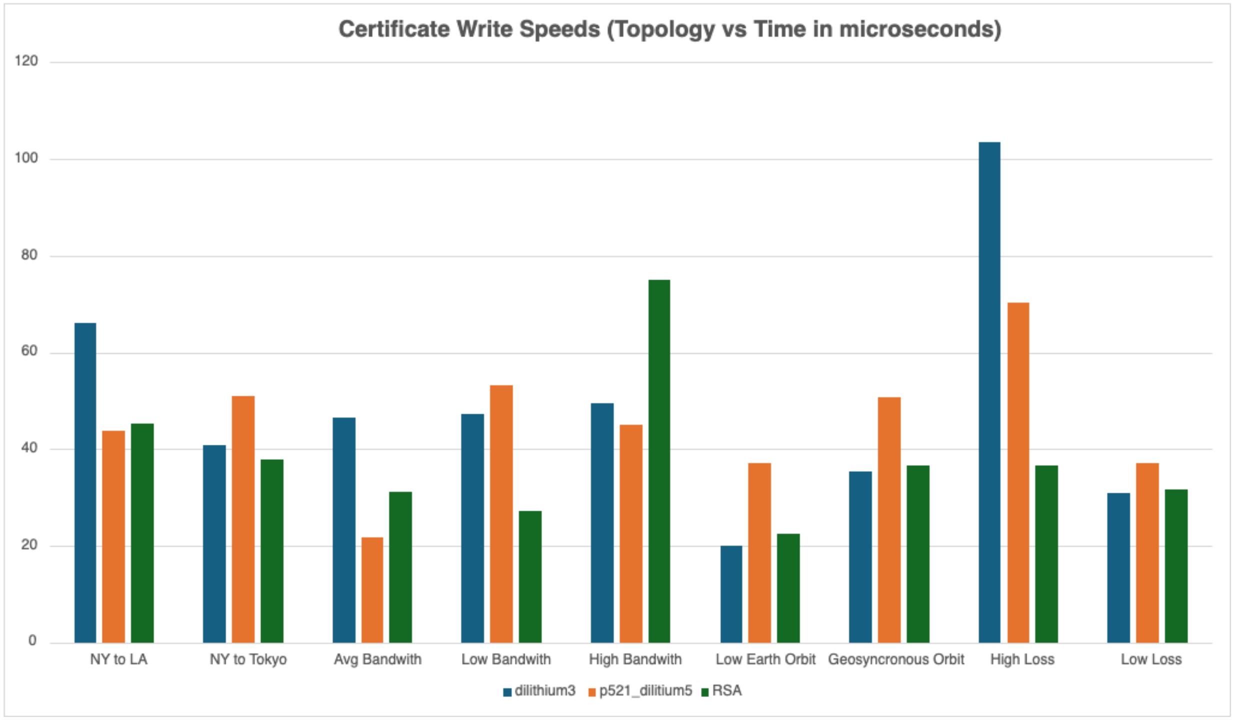
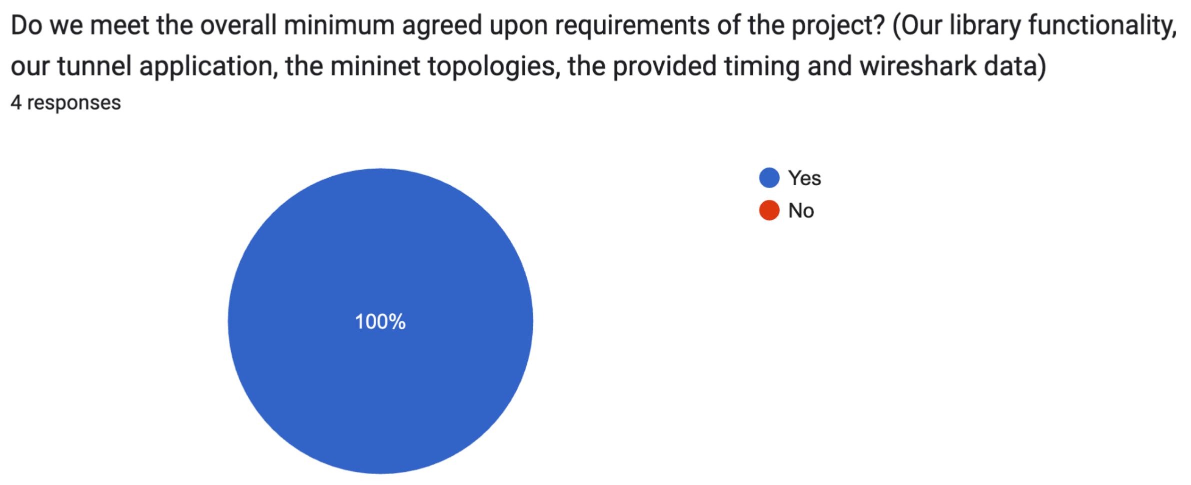


Figure 9.G Certificate Communication Time (Microseconds)

To our surprise, the time it takes to generate an RSA certificate tended to be longer than the time it took to generate our post-quantum certificates. This is likely due to the actual nature of the RSA algorithm and the way the crypto/x509 constructs the certificates. However, the RSA certificates are still much smaller and thus tend to be able to transmit across the network faster than the post-quantum certificates.

9.1 User Acceptance Testing Results

In terms of user acceptance testing, we gave our sponsors access to build materials for running our system along with a survey to record their thoughts on how the system works. We have received positive feedback as well as a couple of areas in which we could improve our system. Overall, it is unanimously agreed upon that we have met the overall minimum requirements of this project, as shown in the feedback in **Figure 9.1.A**.


Do we meet the overall minimum agreed upon requirements of the project? (Our library functionality, our tunnel application, the mininet topologies, the provided timing and wireshark data)

4 responses

Figure 9.1.A Minimum Requirements Met

While this is great news, we also received some constructive feedback from our user testing. They had mentioned inconsistencies in our build materials, typos in different instructions that could be fixed. Moreover, there was an overwhelming report that our timing data was inadequate, this is shown in **Figure 9.1.B** below.

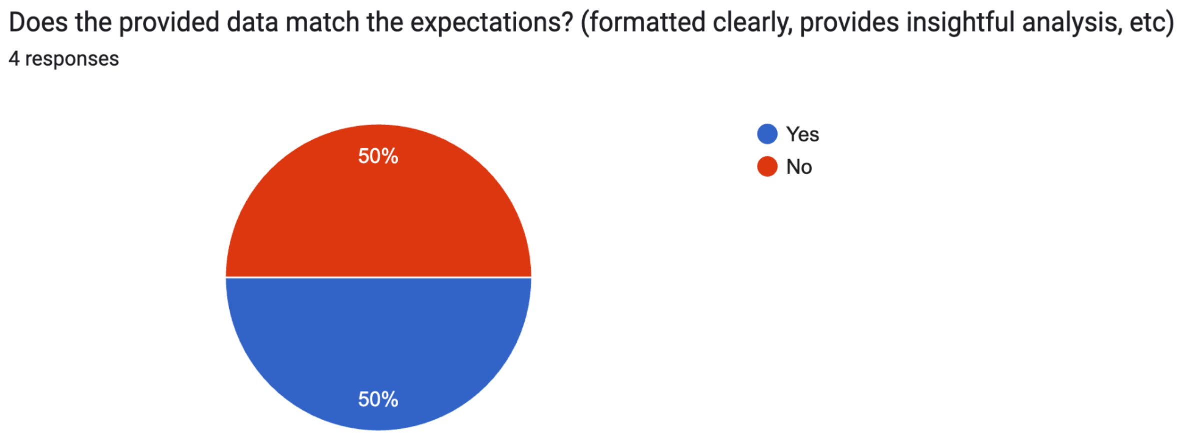


Figure 9.1.B Timing Data Meeting Expectations

We took a closer look at this feedback and looked into the elaboration provided by the users. It turns out that because we were viewing the project from a development perspective and having coded the performance suite, something that was obvious to us was very much not intuitive for a user. The metrics we were providing lacked all units, meaning there was no indication of how the different metrics were really related. This is definitely something that is in need of improvement and can be fixed with a simple code push.

After fixing the problem described above, we received two more survey responses, both of which indicated that our provided data was meeting the expectations of the users.

Lastly, in terms of our UI, we have received very positive feedback, as shown in **Figure 9.1.C** below. We were told that we provided ample instructions on how to interact with our system and that there were only a couple areas we should look into. Starting off, our logging can provide more informative outputs that describe what the client and server are doing at different times.

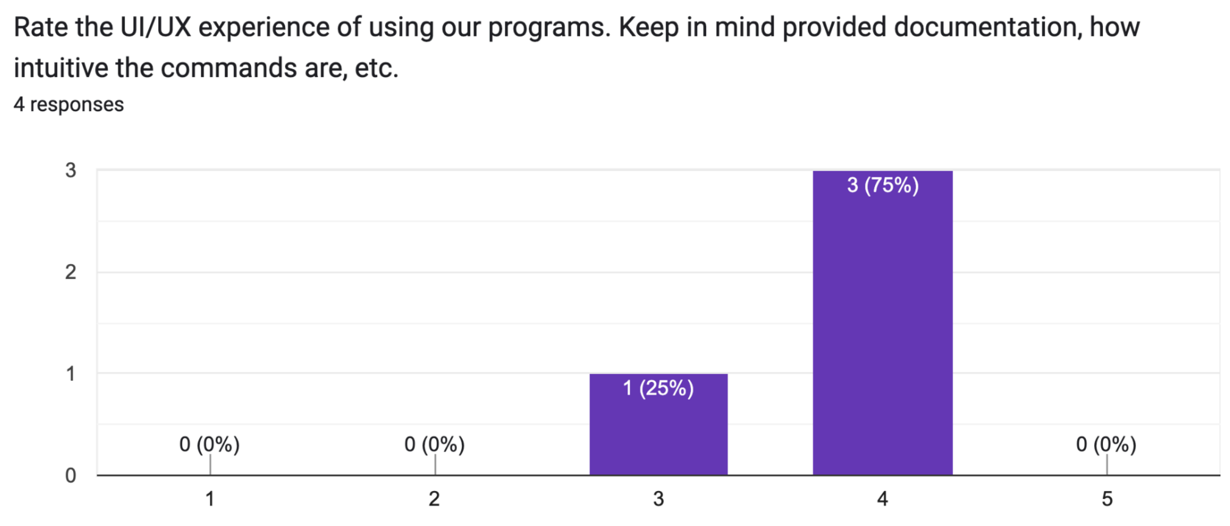


Figure 9.1.C UI/UX Feedback

After collecting the information from the users in our survey, we met with our sponsors to go over the data and talk about our success in the project. The meeting was very productive as we went through each part of the survey and talked about future steps for our project.

In the end, the feedback we received works to show that we have met the definition of success as outlined earlier in this report. By meeting the minimum requirements for this project, our sponsors will be able to access relevant data and a system that can be used to rework and verify that data.

For a more exhaustive list of our results, please visit Annex 8.

10. Discussion

Methodology

This project worked out very well for us in that it allowed us to experiment with quantum-safe cryptography amongst a variety of network topologies. Everything was able to blend together very well in order to create a completely functional tunnel that can be run on a network simulator.

We have been able to test a variety of scenarios, but of course we could always test more topologies. Unfortunately, these different scenarios were all run using a simulation software, Mininet. While great for what we were doing, this means that the real world scenarios may slightly differ as no simulation can 100% mimic the real world.

Had we been given more resources and more time, it would have been amazing to test our client/server application across real networks that stretch the nation.

Project Management

Our project management went through a couple different periods throughout the lifetime of this project. At times, our group functioned like a well oiled machine, outputting assignments of high quality at fast rates. At other times, we became stressed by other classes, large code bases, GitHub merge conflicts, etc. and were unable to turnover as many assignments.

This type of ebb and flow is expected when working with a team, however we definitely missed out on some opportunities to implement certain structures that would have mitigated the ebbs. Starting off, our communication should have been standardized. We should have had defined rules from the start on when we need to communicate certain things to the rest of the team members. There were times when someone may have gotten sick and could not join a meeting. They would notify the team as they thought of it, but sometimes this left us in a position where we were wasting time waiting for everyone.

Having set at the beginning strict rules of notifying at least 24 hours in advance if possible would have given our team more time to adjust for the loss of a member at certain meetings.

Moreover, there were times when our team just wanted to output as much code as possible. This meant we were constantly developing, almost independently as we were all just implementing our assigned tasks. Sometimes, people would catch a bug and get left a bit in the dust while others kept moving forwards. This meant that at some point when we all needed to be back on the same page, we would have to spend a lot of time catching up everyone to the same spot.

We should have created an environment wherein if one person caught a bug that took a certain amount of time to solve, everyone else would turn and help solve the issue. This would’ve kept us more on the same page, we would’ve solved issues faster, and in all would’ve been able to produce code even faster.

Luckily, our team was able to do a couple of things very correctly from the start. Our solution is divided into a couple of different components of which did not rely on each other completely. This gave us an opportunity to split our team of 5 developers into 2 teams of developers specializing in different parts of the project. We were able to simultaneously implement the library along with the client/server application and mininet. This saved us a ton of time and allowed us to become very familiar with the part we were implementing.

Ethics

The solution we have created is expected to become large-scale heading into the future. As of now, quantum computing does not pose much of a threat to cryptography as these computers are not widely available yet. However, they are rapidly increasing in popularity and many companies have begun investing in the development of very large machines.

Once quantum computers reach a point where they start being taken seriously, every single cryptographic application will need to offer support for quantum-safe cryptography, exactly what our solution offers.

There are tons of cases where quantum cryptography can be misused. One of the biggest cases is in data encryption. People can steal data and use quantum cryptography to encrypt it, making it practically impossible to retrieve. They can hold this information for ransom, keep it for themselves, etc.

Moreover, people can create quantum safe networks themselves where they are free to discuss illegal criminal activities. Quantum encryption would make it hard for authorities to track the network and figure out exactly what is going on within the network.

This begs the question then, should we implement some sort of backdoor into the system? Something that can be used by administrators to decrypt quantum-safe data. This would allow us to undo any malicious attacks or uncover any secret criminal networks. While at the surface this may seem like a good idea, implementing a backdoor into cryptography is a huge mistake.

The reason people use cryptography is to keep sensitive information private. The risk of having certain information become public far outweighs the chance that people may use it for illegal activities. A backdoor into a system can be exploited and attackers can do so much more evil when they have access to all data.

11. Future Work

There exists an immediate impact of our project as we are exporting a Go wrapper for OQS X.509, which does not exist. This library is readily available for any Go application that wishes to start producing quantum-safe certificates and verifying them. No more should any package in Go have to be dispatched to a terminal to do so.

If a team wishes to expand the wrapper we have created for OQS X.509, they may want to start by providing functionality for different SDKs. Applications that rely on code bases written in Java, C, Ruby, etc. will struggle with the same problem of not having easy access to certificate generation and verification.

Moreover, the data we provide is very extensive, and can be analyzed by any server administrator to reflect on the trade-off between security and performance requirements. People can quickly look at our data and see results for many different network topologies showing the impact of post-quantum cryptography on a network. They can find situations that apply directly to them and use the data to decide whether or not the extended security is worthwhile at the moment.

There will come a time when quantum computing can not be ignored, and quantum-safe cryptography will need to be implemented in every cryptographic application. Our data will help ease the decision on exactly what needs to be updated and our tunnel application will show how it may look.

There is still much that can be done with our project, should a team wish to continue it. Starting off, we were never able to implement our library in a pre-existing code repository such as OpenZiti. The next steps would be to try forking this repository and connecting different ziti components together with our quantum-safe certificates.

Moreover, we were only able to explore a limited number of different network topologies and different latency/bandwidth configurations within those topologies. Not every network in the world today will apply to the configurations we have data for. A team continuing our work would be able to do more research on configurations that are specific to certain applications and run the tests on these configurations. This would provide the public with even more data on the effect of quantum-safe cryptography over the network.

12. Conclusion

With the completion of this project, we have created a Go wrapper for quantum-safe X.509, created a client/server tunnel which allows for quantum-safe communication, and tested the server on a variety of network topologies. We have found that there is a measurable difference in the performance of networks that utilize quantum-safe cryptography, which is felt much more when the network is in an unideal state, such as when there is high packet loss.

Overall, quantum-safe cryptography is a very viable concept and can be implemented well into many applications all around the world. Most online databases, websites, and applications can seamlessly integrate new algorithms into their cryptography suites and because they operate in a normal networking environment, the performance differences will not be too noticeable. The threat of quantum computing can be subsided as these different algorithms are optimized and mastered going forwards.

Applications that are often used in noisy environments or in environments with limited bandwidth may suffer more with the integration of post-quantum cryptography, but as advances are made every day in quantum computing, there will come a time when security needs to be ensured.

13. References

Aikata, Aikata, et al. "A Crystal for Post-Quantum Security Using Kyber and Dilithium." Research Gate, Jan. 2022, www.researchgate.net/publication/365332761_A_Crystal_for_Post-Quantum_Security_Using_Kyber_and_Dilithium.

An, Hyeongcheol, et al. "Performance Evaluation of Liboqs in Open Quantum Safe Project." The Institute of Electronics, Information and Communication Engineers, 23 Jan. 2018, caislab.kaist.ac.kr/publication/paper_files/2018/SCIS'18_HC_SCA.pdf

Bilogrevic, Igor. Satellite Communications: Internet Challenges and Low-latency Applications. (2008). In I. Bilogrevic (Ed.), Infoscience. https://infoscience.epfl.ch/record/147212

Bindel, N., Braun, J., Gladiator, L., Stöckert, T., & Wirth, J. (2019, August 12). X.509-Compliant Hybrid Certificates for the Post-Quantum Transition. Open Journals. https://www.theoj.org/joss-papers/joss.01606/10.21105.joss.01606.pdf

Bos, J., Ducas, L., & Kiltz, E. (n.d.). Crystals - Kyber: A CCA-secure module-lattice-based Kem | IEEE ... https://ieeexplore.ieee.org/abstract/document/8406610

Cécile Delerablée, & Pointcheval, D. (2008). Dynamic Threshold Public-Key Encryption. Lecture Notes in Computer Science, 317–334. https://doi.org/10.1007/978-3-540-85174-5_18

Dam, D. T., Tran, T. H., Hoang, V. P., Pham, C. K., & Hoang, T. T. (2023). A survey of post-quantum cryptography: Start of a new race. Cryptography, 7(3), 40. https://www.mdpi.com/2410-387X/7/3/40

Debnath, S. K., Choudhury, T., Kundu, N., & Dey, K. (2021, January 29). Post-quantum secure multi-party private set-intersection in Star Network topology. Journal of Information Security and Applications. https://www.sciencedirect.com/science/article/abs/pii/S2214212620308668

Gergely, A. M., & Crainicu, B. (1970, January 1). BlockCACert – a blockchain-based novel concept for automatic deployment of X.509 Digital certificates. SpringerLink. https://link.springer.com/chapter/10.1007/978-3-030-93817-8_73

H. Haddadi, M. Rio, G. Iannaccone, A. Moore and R. Mortier, "Network topologies: inference, modeling, and generation," in IEEE Communications Surveys & Tutorials, vol. 10, no. 2, pp. 48-69, Second Quarter 2008, doi: 10.1109. https://ieeexplore.ieee.org/abstract/document/4564479

Irei, A., & Shea, S. (2022, October 20). What is the zero-trust security model?. TechTarget Security. https://www.techtarget.com/searchsecurity/definition/zero-trust-model-zero-trust-network

Jena, B. K. (2023, February 9). What is AES encryption and how does it work?. Simplilearn.com. https://www.simplilearn.com/tutorials/cryptography-tutorial/aes-encryption

Jøsang, A., Zych, M. D., Vishi, K., & Mavroeidis, V. (2018, March 31). The Impact of Quantum Computing on Present Cryptography. Cornell University. Retrieved February 8, 2024, from https://arxiv.org/abs/1804.00200

Kampanakis, P., Panburana, P., Daw, E., & Geest, D.V. (2018). The Viability of Post-quantum X.509 Certificates. IACR Cryptol. ePrint Arch., 2018, 63. https://eprint.iacr.org/2018/063.pdf

Laboratories, G. J. S. S., Simmons, G. J., Laboratories, S., & Metrics, O. M. A. (1979b, December 1). Symmetric and asymmetric encryption. ACM Computing Surveys. https://dl.acm.org/doi/abs/10.1145/356789.356793

Lu, C.-C., & Tseng, S.-Y. (2002, November 7). Integrated Design of AES (advanced encryption standard ... IEEE Explore. https://ieeexplore.ieee.org/abstract/document/1030726

Mathur, N., & Bansode, R. (2016). AES Based Text Encryption Using 12 Rounds with Dynamic Key Selection. Procedia Computer Science, 79, 1036–1043. https://doi.org/10.1016/j.procs.2016.03.131

Mohamed, N. N., Othman, H., Mat Isa, M. A., & Hashim, H. (2017, October 19). A secure communication in location based services ... https://www.researchgate.net/publication/320653391_A_secure_communication_in_location_based_services_using_AES256_encryption_scheme

Moody, D. (2023, September 27). CSRC presentation: And then there were Four: The first NIST PQC standards. National Institute of Standards and Technology. https://csrc.nist.gov/presentations/2023/mpts2023-day2-talk-nist-pqc-first-standards

NIST announces first four quantum-resistant cryptographic algorithms. NIST. (2022, July 7). https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms

Quantum-readiness: Migration to post-quantum cryptography: CISA. Cybersecurity and Infrastructure Security Agency. (2023, August 17). https://www.cisa.gov/resources-tools/resources/quantum-readiness-migration-post-quantum-cryptography

R. L. S. de Oliveira, C. M. Schweitzer, A. A. Shinoda and Ligia Rodrigues Prete, "Using Mininet for emulation and prototyping Software-Defined Networks," 2014 IEEE Colombian Conference on Communications and Computing (COLCOM), Bogota, Colombia, 2014, pp. 1-6, doi: 10.1109. https://ieeexplore.ieee.org/abstract/document/6860404

R. R. Fontes, S. Afzal, S. H. B. Brito, M. A. S. Santos and C. E. Rothenberg, "Mininet-WiFi: Emulating software-defined wireless networks," 2015 11th International Conference on Network and Service Management (CNSM), Barcelona, Spain, 2015, pp. 384-389, https://doi.org/10.1109/CNSM.2015.7367387

Rao Sandeep, Mahto Dindayal, Yadav Dilip, Khan Danish. The AES-256 Cryptosystem Resists Quantum Attacks. (2017). National Institute of Technology. https://www.researchgate.net/profile/Sandeep-Rao-4/publication/316284124_The_AES-256_Cryptosystem_Resists_Quantum_Attacks/links/58f999200f7e9ba3ba4d22b1/The-AES-256-Cryptosystem-Resists-Quantum-Attacks.pdf?_sg%5B0%5D=started_experiment_milestone&origin=journalDetail&_rtd=e30%3D

Reimair, F., Feichtner, J., & Teufl, P. (2015, October 23). IEEE Xplore. https://ieeexplore.ieee.org/Xplore/home.jsp

Stebila, Douglas, and Michele Mosca. "Post-Quantum Key Exchange for the Internet and the Open Quantum Safe Project." SpringerLink, Springer International Publishing, 20 Oct. 2017, link.springer.com/chapter/10.1007/978-3-319-69453-5_2#Sec14.

Annex 1: Project Plan

Our group will be operating out of an agile process model. With this structure, we will be operating in "coding sprints" wherein we define a set of actionable tasks over a short period of time. During the sprint, any actionable tasks will be assigned to members specifically for them to complete by the next sprint. Each sprint will lead to a Minimum Viable Product (MVP). A MVP should be a working iteration of our project, although may be relatively bare compared to the fully implemented project. Every coding sprint builds off of the previous sprint and after all of our sprints have been completed, we will be left with a full, working product.

A1.1 Implementation Schedule

The project will be broken down into 10 coding sprints, each of which will last one week. During each sprint, action items will be assigned to group members to complete by the next sprint.

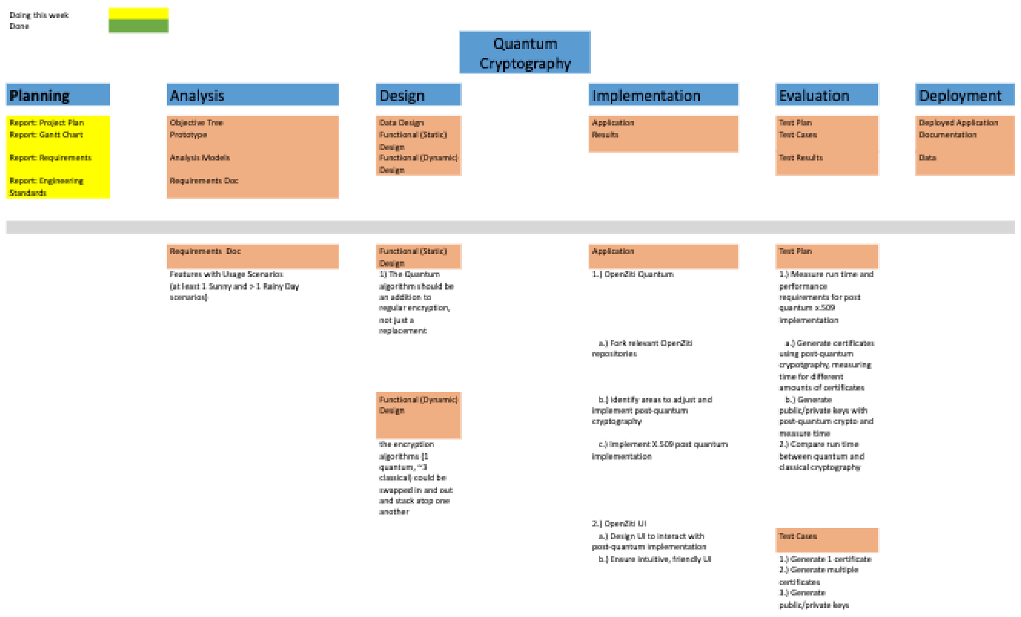
Our project has been designed to go through six different phases, in which we will be conducting different work to ensure the quality and integrity of our project. The six phases are as follows:

1. Planning
    - Determining the various requirements, standards, and overall plan for our project
2. Analysis
    - Discussing and documenting the various usage scenarios our project must be able to accomplish and modeling different objectives
3. Design
    - Functional and data designs to establish a method of implementing all relevant parts of the project, a way to store or handle any data, handle the overall flow from component to component
4. Implementation
    - Developing a Go library for the Open Quantum-Safe (OQS) framework to provide post-quantum X.509 functionality for Go applications.
    - Creating a Client/Server Tunneling Application to demonstrate and test the integration of the Go OQS library.
    - Setting up Mininet for network simulation to test the application in various network topologies.
5. Evaluation
    - Measuring the performance of the post-quantum X.509 system in terms of speed, security, and compatibility.
    - Comparing the post-quantum implementation with classical cryptographic methods to assess improvements and potential drawbacks.
6. Deployment

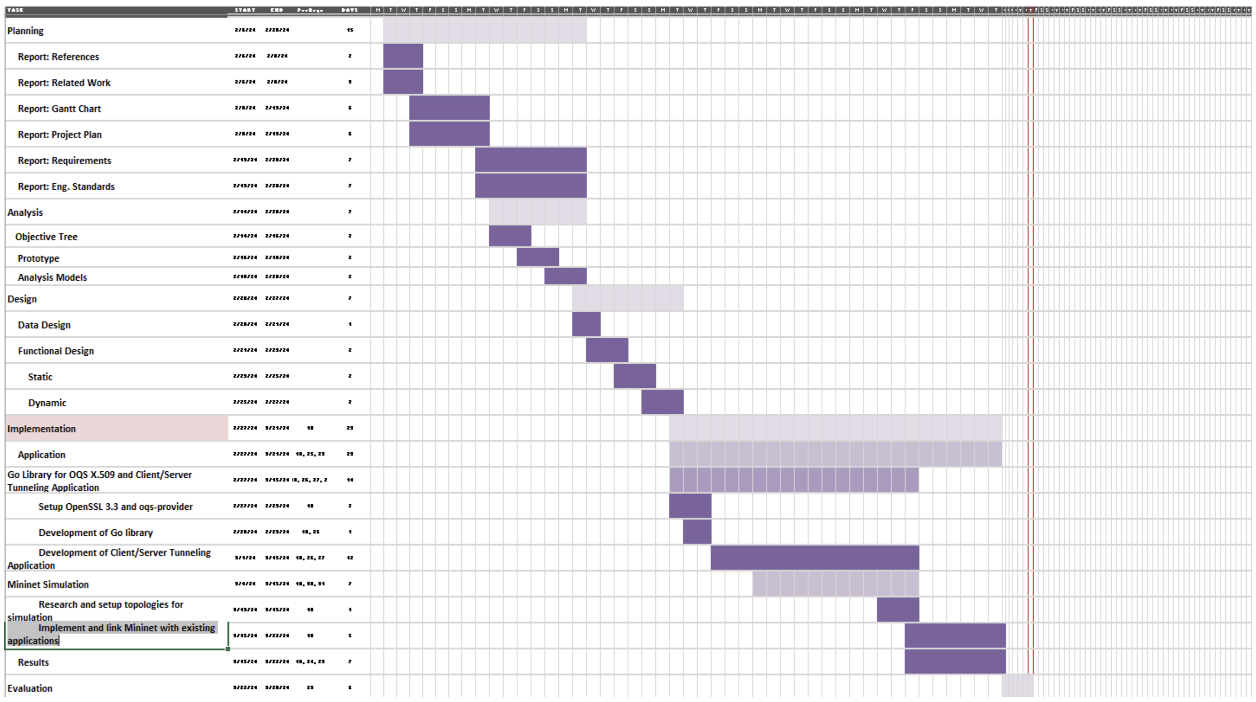
- ○ Considering the integration of the system into broader applications or repositories as future work.
- ○ Finalizing the implementation, ensuring proper documentation, and preparing for the presentation at the COE Showcase.

The completion of each phase is critical to the success of the following phases, and so we have outlined what tasks need to be completed during each phase and by when these tasks should be complete.

The following Work Breakdown Structure (WBS) shows how each phase of the project breaks down and what items should be completed during the phase.


Doing this week
Done
Quantum Cryptography
Planning
Report: Project Plan
Report: Gantt Chart
Report: Requirements
Report: Engineering Standards
Analysis
Objective Tree
Prototype
Analysis Models
Requirements Doc
Design
Data Design
Functional (Static) Design
Functional (Dynamic) Design
Implementation
Application
Results
Evaluation
Test Plan
Test Cases
Test Results
Deployment
Deployed Application
Documentation
Data
Requirements Doc
Features with Usage Scenarios (at least 1 Sunny and > 1 Rainy Day scenarios)
Functional (Static) Design
1) The Quantum algorithm should be an addition to regular encryption, not just a replacement
Functional (Dynamic) Design
the encryption algorithms (1 quantum, ~3 classical) could be swapped in and out and stack atop one another
Application
1.) OpenZiti Quantum
a.) Fork relevant OpenZiti repositories
b.) Identify areas to adjust and implement post-quantum cryptography
c.) Implement X.509 post quantum implementation
2.) OpenZiti UI
a.) Design UI to interact with post-quantum implementation
b.) Ensure intuitive, friendly UI
Test Plan
1.) Measure run time and performance requirements for post quantum x.509 implementation
a.) Generate certificates using post-quantum crypotgraphy, measuring time for different amounts of certificates
b.) Generate public/private keys with post-quantum crypto and measure time
2.) Compare run time between quantum and classical cryptography
Test Cases
1.) Generate 1 certificate
2.) Generate multiple certificates
3.) Generate public/private keys

Additionally, given deadlines set by the class schedule, we have weighted and estimated the time it would take to complete each phase. The tasks and their estimated completion dates are tracked in the GANTT chart below.



TASK	START	END	PreReqs	DAYS
Planning	2/6/24	2/20/24		15
Report: References	2/6/24	2/8/24		2
Report: Related Work	2/6/24	2/8/24		3
Report: Gantt Chart	2/8/24	2/13/24		6
Report: Project Plan	2/8/24	2/13/24		6
Report: Requirements	2/13/24	2/20/24		7
Report: Eng. Standards	2/13/24	2/20/24		7
Analysis	2/14/24	2/20/24		7
Objective Tree	2/14/24	2/16/24		2
Prototype	2/16/24	2/18/24		2
Analysis Models	2/18/24	2/20/24		2
Design	2/20/24	2/27/24		7
Data Design	2/20/24	2/21/24		1
Functional Design	2/21/24	2/23/24		2
Static	2/23/24	2/25/24		2
Dynamic	2/25/24	2/27/24		2
Implementation	2/27/24	3/21/24	18	23
Application	2/27/24	3/21/24	18, 25, 23	23
Go Library for OQS X.509 and Client/Server Tunneling Application	2/27/24	3/15/24	18, 26, 27, 2	14
Setup OpenSSL 3.3 and oqs-provider	2/27/24	2/29/24	18	2
Development of Go library	2/28/24	2/29/24	18, 26	1
Development of Client/Server Tunneling Application	3/1/24	3/15/24	18, 26, 27	12
Mininet Simulation	3/4/24	3/15/24	18, 30, 31	7
Research and setup topologies for simulation	3/13/24	3/15/24	18	1
Implement and link Mininet with existing applications	3/15/24	3/22/24	18	5
Results	3/15/24	3/22/24	18, 24, 25	7
Evaluation	3/22/24	3/28/24	23	6

The estimated completion of the Evaluation phase is Mar. 28th. At this point, the group can focus on presenting the solution to various showcases, stakeholders, etc.

A1.2 Division of Labor and Responsibilities

Our group is operating as a 5-manned team, each person having different and essential roles. We have divided into the following 5 positions:

- Scope – Shifan Hirani
  - Ensuring all aspects of the project are well defined and within reasonable, established time frames.
- Schedule – Josef Munduchirakal
  - Ensuring we are on time to complete all deadlines, outlining actionable tasks, etc.
- Stakeholder Management / Communication – Blake Lun
  - Conducting all communication with the stakeholders, making sure we meet all their needs and expectations, reaching out with any questions
- Risks – Grant Shields
  - Makes sure any team dynamics issues or stakeholder issues are addressed and resolved
- Quality – Aidan Heffron
  - Ensuring all output exceeds expectations and is free of any errors, mistakes, or incomplete information

While operating at each members' individual position in the team, of course we will have actionable tasks coming from a backlog during each sprint and assigned to each team member. Following are the sprints, split into 3 eras, namely Pre-Development, Development, and Post-Development. (1 point equates to ½ day of development)

Pre-Development

Sprint 1: Feb. 6 – Feb. 13

Task	Assigned To	Estimated Points
Gantt Chart	All	2
Project Plan	All	4
References	All	2
Related Work	All	2
Checkpoint 1	All	4

Sprint 2: Feb. 13 – Feb. 20

Task	Assigned To	Estimated Points
Requirements	All	2
Eng. Standards	All	2
Checkpoint 2	All	4
Clone OpenZiti	All	1
Ramp up OpenZiti system	All	2

Sprint 3: Feb. 20 – Feb. 27

Task	Assigned To	Estimated Points
Data Design	Josef, Grant	2

Functional Static Design	Shifan	2
Functional Dynamic Design	Aidan, Blake	2
Follow OpenZiti first demo	All	2

Development

Sprint 4: Feb. 27 – Mar. 5

Task	Assigned To	Estimated Points
Clone OpenZiti C SDK	Aidan, Blake, Josef, Shifan	1
Identify relevant files for enrolling process	Aidan, Shifan	2
Identify relevant UI files	Blake, Josef	2
Clone PQ X.509 repo	Grant	1
Begin exporting PQ implementation to implement in OpenZiti	Grant	2

Sprint 5: Mar. 5 – Mar. 12

Task	Assigned To	Estimated Points
Integrate PQ into openZiti for key gen	Grant, Aidan, Shifan	6
Adjust console line UI to account for PQ key gen options	Blake, Josef	4
Setup cloud env for testing performance	Blake Josef	2

Sprint 6: Mar. 12 – Mar. 19

Task	Assigned To	Estimated Points
Integrate PQ into openZiti for cert gen	Grant, Aidan, Shifan	6
Adjust console line UI to account for PQ cert options	Blake, Josef	4
Clone updated OpenZiti to cloud resources for testing	Blake, Josef	4

Sprint 7: Mar. 19 – Mar. 26

Task	Assigned To	Estimated Points
Setup Go Environment	All	2
Explore OQS Library	All	2
Begin Development of Go OQS Library	Grant, Aidan, Shifan	4
Setup Basic Client/Server Tunneling App	Blake, Josef	2

Sprint 8: Mar 26. – Apr. 2

Task	Assigned To	Estimated Points
Continue Development of Go OQS Library	Grant, Aidan, Shifan	4
Integrate Go OQS Library into Tunneling App	Grant, Aidan, Shifan	4
Setup Mininet for Network Simulation	All	2

Sprint 9: Apr. 2 – Apr. 9

Task	Assigned To	Estimated Points
Finalize Go OQS Library	Grant, Aidan, Shifan	4
Test Key Exchange and Encryption	Blake, Josef	4
Run Network Simulations in Mininet	All	4

Post-Development

Sprint 10: Apr. 9 – Apr. 16

Task	Assigned To	Estimated Points
MVP Deadline	All	8
Presentation Slides	All	8
COE Showcase Prep	All	8

A1.3 Budget Costs

Internal Course Resource Needs

For our project, we will require a working budget of $0. We will be utilizing open-source libraries and tools, such as Open Quantum-Safe (OQS) and Mininet, which are freely available. This library prides itself on being easily accessible to consuming applications, and beyond that, free. The primary components of our project are:

- Go Library for OQS X.509: This library will be developed to provide Go functions that wrap around OQS commands, allowing them to be used by Go applications. The setup requires the installation of OpenSSL 3.3 and oqs-provider, which are also open-source.
- Client/Server Tunneling Application: This application will demonstrate the use of our qs509 library for post-quantum X.509 authentication, key exchange, and AES256 encryption. It will serve as a proof of concept for the implementation of post-quantum cryptography in network communication.
- Mininet Simulation: We will use Mininet to simulate different network topologies for testing our client/server programs. Mininet is a free software that provides pre-loaded virtual machine images.

The total list of resources required for our internal course project is:

- A laptop capable of running a Virtual Machine, a Bash Environment, and Go development tools – $0
- Open Quantum-Safe (OQS) – $0
- OpenSSL 3.3 – $0
- oqs-provider – $0
- Mininet – $0

Overall Cost of Project to an Outside Company

Expanding the project beyond the classroom and adapting it for commercial use involves additional considerations. These include licensing, manpower for development and implementation, and extensive testing.

- Licensing - If a company decides to use a licensed version of post-quantum cryptography solutions for enhanced security or compliance reasons, licensing costs could be significant. Estimated cost: Up to $1,000 per year (varies based on the solution).
- Implementation - The expansion of the project to include Go library development, client/server applications, and network simulation requires significant developer effort.

Assuming a team of 5 engineers working for 3 months, the cost based on an average annual salary of $80,000 would be approximately $100,000.

- Maintenance and Updates - While maintenance costs remain low due to the nature of the project, periodic updates to algorithms and libraries may be necessary. Estimated cost: Minimal to moderate, depending on the frequency and complexity of updates.
- Testing Infrastructure - Testing the system would be much more in depth for a company's iteration of the project. Rather than testing via separate virtual hosts, a company may elect to build the network with actual machines with different performance requirements, located in different areas of the globe. These machines themselves would cost more and the facilities required to host them in separate regions would cost as well. Estimates for this can be expected to be in the thousands, likely anywhere from $5,000 to $30,000 depending on the testing the company may want to conduct.

Given the enhanced project scope and the potential for commercial application, a company could expect to spend upwards of $105,000 to $130,000 for a fully implemented system.

A1.4 Quality Plan

Given the critical nature of our project, which involves quantum-safe cryptography and network security, maintaining high-quality standards is imperative. Our goal is to ensure that the implementation of the Go library for OQS X.509, the client/server tunneling application, and the Mininet simulation meets the expectations of stakeholders and integrates seamlessly with the existing OpenZiti repository. To achieve this, we have outlined a comprehensive quality plan with the following objectives:

1. Descriptive and Understandable Code: All code should be self-explanatory to a programmer with moderate familiarity with the language and domain. This includes meaningful variable names, function names, and comments that clarify the purpose and flow of the code.
2. Consistent Formatting and Structure: The code should adhere to the formatting and structural conventions of the OpenZiti repository. This consistency is crucial for ease of integration and maintenance.
3. Robust and Reliable Functionality: The core logic, especially related to encryption and network security, must be foolproof. There should be no vulnerabilities, memory leaks, or potential points of failure.

To ensure these objectives are met, we have devised the following plan:

- Automated Linting: Utilize linting tools to automatically check and enforce coding standards and formatting consistency across the codebase.
- Code Reviews: Conduct regular code reviews with the development team and stakeholders. These reviews will focus on evaluating the functionality, naming

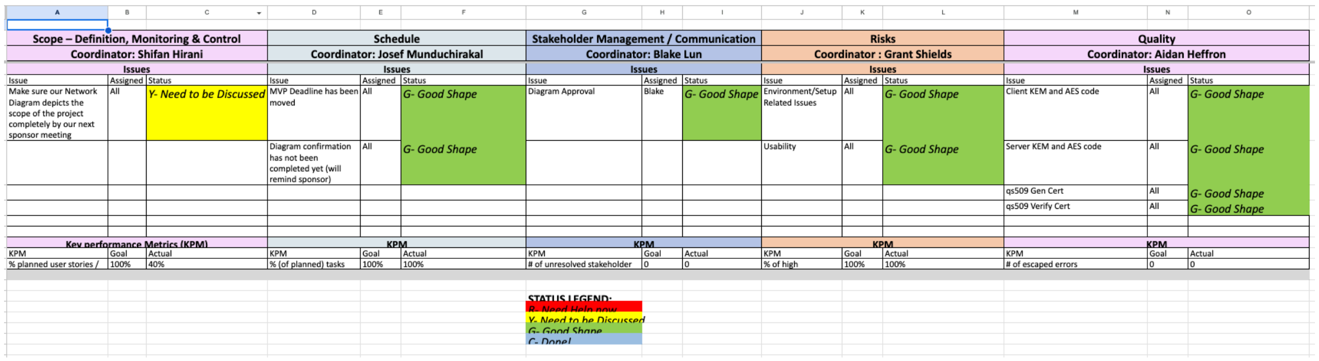
conventions, and overall design of the code. Feedback from these sessions will be used to refine and improve the code.

- Comprehensive Documentation: Create detailed documentation for all functions, modules, and components. This documentation should explain the purpose, inputs, outputs, and usage examples for each element of the code.
- Extensive Testing: Develop a thorough testing strategy that includes unit tests, integration tests, and end-to-end tests. These tests should cover all critical paths and edge cases to ensure the reliability and security of the system.
- Performance Benchmarking: Conduct benchmarking tests to evaluate the performance of the system under various network conditions and loads. This will help identify any potential bottlenecks or inefficiencies.

By adhering to this quality plan, we aim to deliver a robust and reliable solution that meets the high standards required for network security and post-quantum cryptography.

Annex 2: Project Management Artifacts

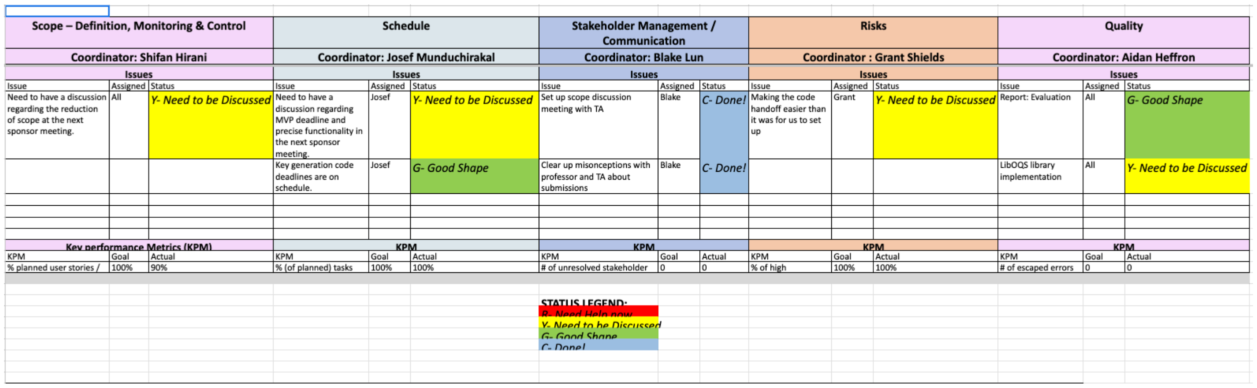
Our group utilized weekly meetings to discuss items that went well, went bad, could be improved, etc. These meetings all resulted in "Group Updates" of which highlighted our status each week. A couple of these reports shown below:



Scope – Definition, Monitoring & Control			Schedule			Stakeholder Management / Communication			Risks			Quality		
Coordinator: Shifan Hirani			Coordinator: Josef Munduchirakal			Coordinator: Blake Lun			Coordinator : Grant Shields			Coordinator: Aidan Heffron		
Issues			Issues			Issues			Issues			Issues		
Issue	Assigned	Status	Issue	Assigned	Status	Issue	Assigned	Status	Issue	Assigned	Status	Issue	Assigned	Status
Make sure our Network Diagram depicts the scope of the project completely by our next sponsor meeting	All	*Y- Need to be Discussed*	MVP Deadline has been moved	All	*G- Good Shape*	Diagram Approval	Blake	*G- Good Shape*	Environment/Setup Related Issues	All	*G- Good Shape*	Client KEM and AES code	All	*G- Good Shape*
			Diagram confirmation has not been completed yet (will remind sponsor)	All	*G- Good Shape*				Usability	All	*G- Good Shape*	Server KEM and AES code	All	*G- Good Shape*
												qs509 Gen Cert	All	*G- Good Shape*
												qs509 Verify Cert	All	*G- Good Shape*
Key performance Metrics (KPM)			KPM			KPM			KPM			KPM		
KPM	Goal	Actual	KPM	Goal	Actual	KPM	Goal	Actual	KPM	Goal	Actual	KPM	Goal	Actual
% planned user stories /	100%	40%	% (of planned) tasks	100%	100%	# of unresolved stakeholder	0	0	% of high	100%	100%	# of escaped errors	0	0

STATUS LEGEND:
R- Need Help now
Y- Need to be Discussed
G- Good Shape
C- Done!

Figure 2.A Group Update 7



Scope – Definition, Monitoring & Control			Schedule			Stakeholder Management / Communication			Risks			Quality		
Coordinator: Shifan Hirani			Coordinator: Josef Munduchirakal			Coordinator: Blake Lun			Coordinator : Grant Shields			Coordinator: Aidan Heffron		
Issues			Issues			Issues			Issues			Issues		
Issue	Assigned	Status	Issue	Assigned	Status	Issue	Assigned	Status	Issue	Assigned	Status	Issue	Assigned	Status
Need to have a discussion regarding the reduction of scope at the next sponsor meeting.	All	*Y- Need to be Discussed*	Need to have a discussion regarding MVP deadline and precise functionality in the next sponsor meeting.	Josef	*Y- Need to be Discussed*	Set up scope discussion meeting with TA	Blake	*C- Done!*	Making the code handoff easier than it was for us to set up	Grant	*Y- Need to be Discussed*	Report: Evaluation	All	*G- Good Shape*
			Key generation code deadlines are on schedule.	Josef	*G- Good Shape*	Clear up misconceptions with professor and TA about submissions	Blake	*C- Done!*				LibOQS library implementation	All	*Y- Need to be Discussed*
Key performance Metrics (KPM)			KPM			KPM			KPM			KPM		
KPM	Goal	Actual	KPM	Goal	Actual	KPM	Goal	Actual	KPM	Goal	Actual	KPM	Goal	Actual
% planned user stories /	100%	90%	% (of planned) tasks	100%	100%	# of unresolved stakeholder	0	0	% of high	100%	100%	# of escaped errors	0	0

STATUS LEGEND:
R- Need Help now
Y- Need to be Discussed
G- Good Shape
C- Done!

Figure 2.B Group Update 5

Annex 2.1: Stakeholder Management Plan

Stakeholder Name	Category	Levels of power and interest	Strategies for gaining support / reducing obstacles	Information needed / Document Name	Document Format / Medium (e.g., hardcopy, email, chat, etc.)	How Often / When Due	Status (e.g., stakeholder issues, status - happy or not, etc.)

Customer	Internal	High Power and High Interest	Meet with regularly	Progress updates	Email	Once a week	Happy - Content with all the checkpoints and approves of the scope adjustment
Instructor	Primary	High Power and High Interest	Report to regularly	Progress updates	Document submission through Microsoft Teams	Twice a week	Happy - Checks with our customer to ensure that everything is being met, which it is. Approves of our scope adjustment.
Teaching Assistant	Primary	High Power and High Interest	Report to regularly	Progress updates	Document submission through Microsoft Teams	Twice a week	Happy - Has met with us multiple times to check on progress and likes what they see. Approves of our scope adjustment
Project Team	Primary	High Power and High Interest	Meet weekly to work	All documents work on collectively	Email, chat	Meet as needed, as our MVP date approaches, meet every other day	Happy - Team feels that we are on course and able to accomplish the goal we have set out

Annex 2.2: Risk Management Plan

Additional Notes from headers

- Impact - On a 1-5 scale, where 5 has a big negative impact
- Monitoring Plan - How we determine if the risk is becoming a reality
- Management Plan - How we aim to minimize the negative impact, now that the risk is becoming reality

Risk	**Prob in %**	**Impact**	**Risk Mitigation Plan (plan for avoiding)**	**Risk Monitoring Plan**	**Risk Management Plan**	**Status**
Incomplete code by MVP deadline	15%	5	Communication with stakeholders to identify scope and deadline adjustment if necessary	Weekly updates allow for communication	After spring break, daily group-code meetings	Under Control
Difficult Code Handoff	40%	3	Attempting to create a docker image as a final code submission to reduce environment issues	Developers will become more comfortable with the software and be able to better explain its use	keeping an eye out for easier methods to interact with OpenZiti	Manageable
Issues with certificate creation	10%	3	Test certificates that are generated as soon as possible.	Update the status of generated certificates after spring break.	Observe the performance of custom generated certificates.	Under Control
Code crumbles under stress-testing	10%	4	Testing multiple network topologies with realistic obstacles such as extreme latency	record the performance on each unit test performed	Find the greatest time-sinks and hangups within the code using unit test data	Under Control

Annex 3: User Stories / Usage Scenarios

Priority	User Story	Persona	Acceptance Criteria	Story Points
1	As a web admin, I want to be able to import quantum safe X.509 library into my package	web admin	I can start seeing functions after a simple "import" statement in my program	2
1	As a web admin, I want to be able to quickly set up the quantum safe X.509 library	web admin	I can start using the imported library after a quick build instruction or running a script that builds necessary library components	4
2	As a web admin, I want to be able to call quantum-safe X.509 library to generate a certificate	web admin	I can call a generate certificate function and get returned a valid certificate	4
3	As a web admin, I want to be able to call quantum-safe X.509 library to verify a certificate	web admin	I can call a verify certificate function and determine if a certificate is valid	4
4	As a web admin, I want to be able to call quantum-safe X.509 library to parse a certificate	web admin	I can call a parse certificate function and get returned accurate information about the certificate	4
5	As a web admin, I want to see how long it would take to generate a quantum-safe certificate	web admin	I can call a timer function to measure the time it takes to generate my certificates	3
6	As a web admin, I want to see how long it would take to verify a quantum-safe certificate	web admin	I can call a timer function to measure the time it takes to verify different certificates	3
7	As a web admin, I want to see how long it would take to parse a quantum-safe certificate	web admin	I can call a timer function to see how long it takes to parse different certificates	3
8	As a web admin, I want to see how large a quantum-safe certificate is	web admin	I can call a function to return byte information about my certificates	2

9	As a web admin, I want to see how long it takes to transfer a quantum-safe certificate to my clients	web admin	I can see relevant wireshark data that tracks how long it takes to transfer a certificate	3
10	As a web admin, I want to see how long it takes to generate a quantum-safe key	web admin	I can call a timer function to determine how long it takes to generate quantum-safe keys	3
11	As a web admin, I want to see how large a quantum-safe key is	web admin	I can call a function to print byte data for quantum-safe keys	3
12	As a client, I want to be able to verify a quantum-safe certificate	client	I can call a function that determines if a certificate is valid	2
13	As a client, I want access to documentation on how to build quantum-safe components	client	I can access manual pages for the quantum-safe library build	4
14	As a client, I want to be able to securely communicate with a server that requires quantum-safe cryptography	client	I can use the quantum-safe library to authenticate with servers	4
15	As a client, I want to be able to generate quantum-safe components on my machine	client	I can check performance requirements for the quantum-safe library and determine if a personal machine is capable of what I need	4
16	As a web admin, I want to know what topologies may delay communication on my quantum-safe network	web admin	I have access to a Mininet simulation of different topologies that apply to my situation	6
17	As as web admin, I want access to documentation for all quantum-safe library functions	web admin	I can access manual pages for all quantum-safe library functions	4
18	As a web admin, I want easy to call go functions for all library functions	web admin	I can generate necessary components with as little information as needed by the library and a simple Go function call	3
19	As a web admin, I want options on what	web admin	I can use optional	6

	cryptography algorithms I can use for my network		parameters to specify exactly what signing or KEM algorithm I want to use to authenticate and encrypt communications to users with	
20	As a client, I want to be able to generate cryptographic components used by a server I want to connect to	client	I can see what a server requires and use optional parameters to match the server requirements for all cryptographic components	3
21	As a client, I want to know that the server I connect to is more than just a middle-man	client	I can see the signature on the server certificate	3

Annex 4: Requirements Models

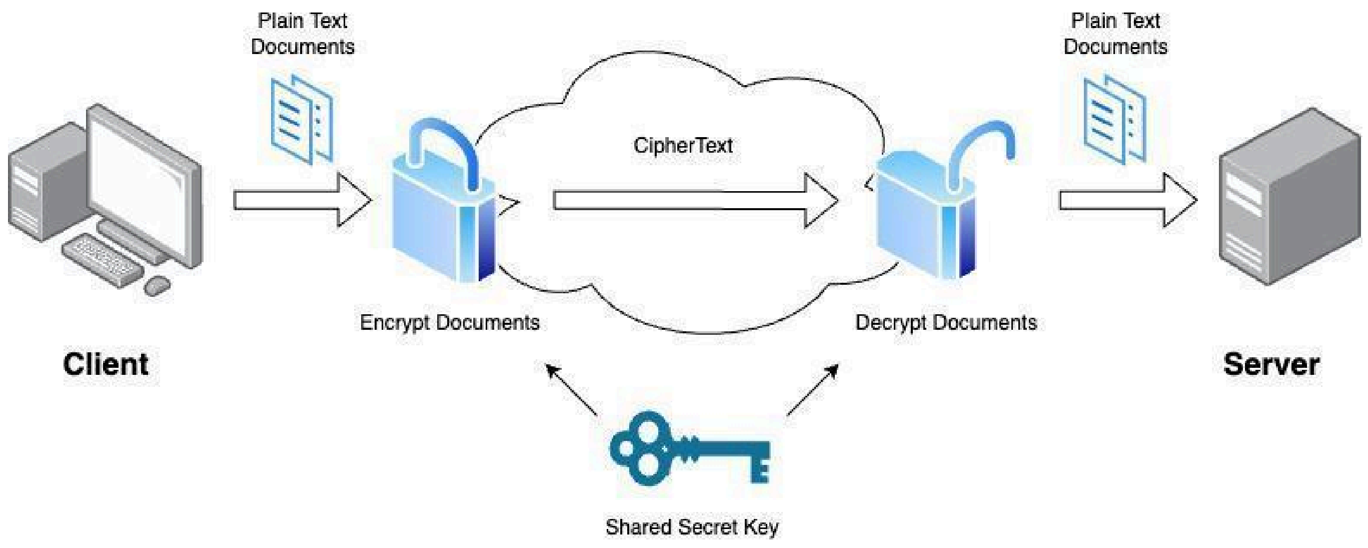


Figure 4.A Client/Server Requirement Model

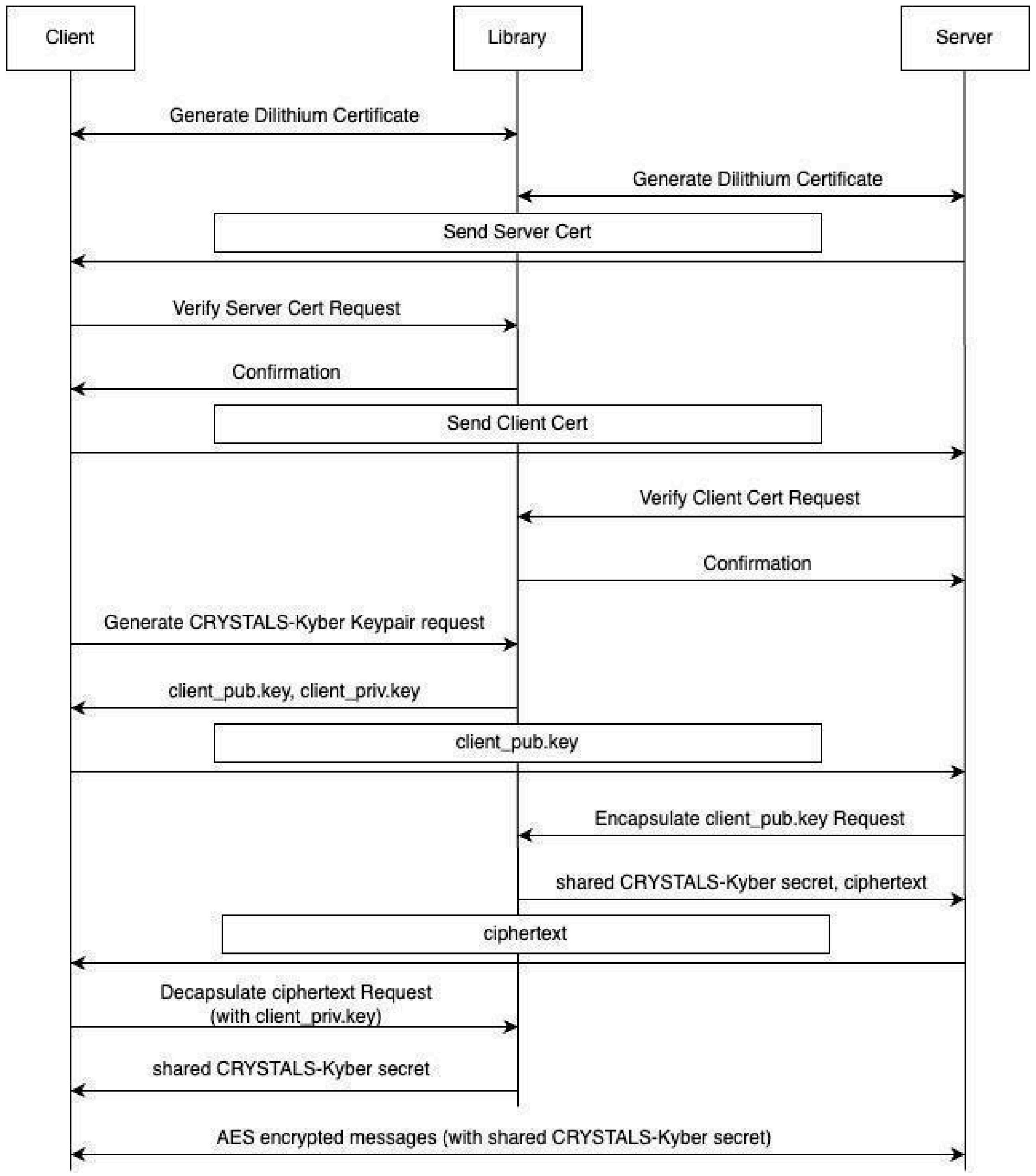
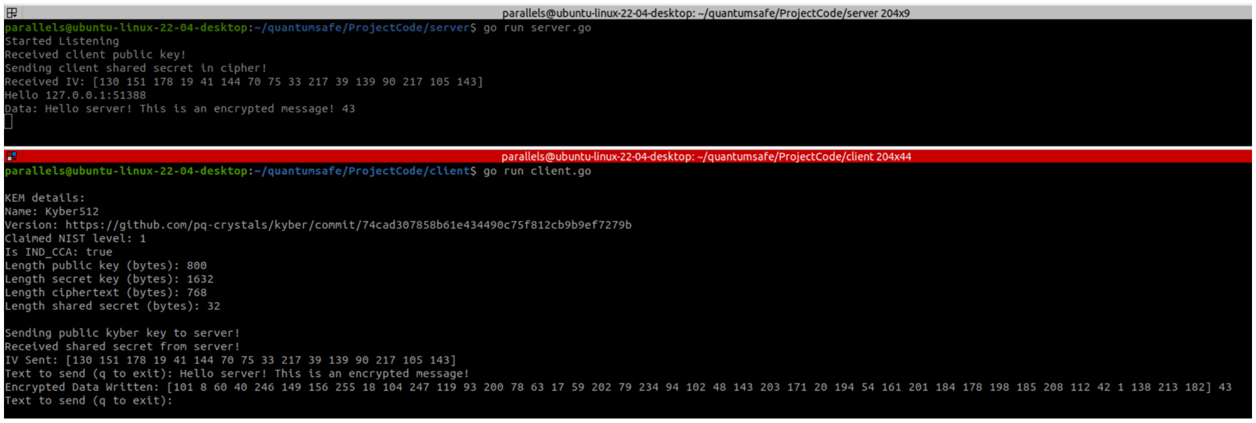

Client
Library
Server
Generate Dilithium Certificate
Generate Dilithium Certificate
Send Server Cert
Verify Server Cert Request
Confirmation
Send Client Cert
Verify Client Cert Request
Confirmation
Generate CRYSTALS-Kyber Keypair request
client_pub.key, client_priv.key
client_pub.key
Encapsulate client_pub.key Request
shared CRYSTALS-Kyber secret, ciphertext
ciphertext
Decapsulate ciphertext Request (with client_priv.key)
shared CRYSTALS-Kyber secret
AES encrypted messages (with shared CRYSTALS-Kyber secret)

Figure 4.B Client/Server Sequence Requirements

Annex 5: Prototype



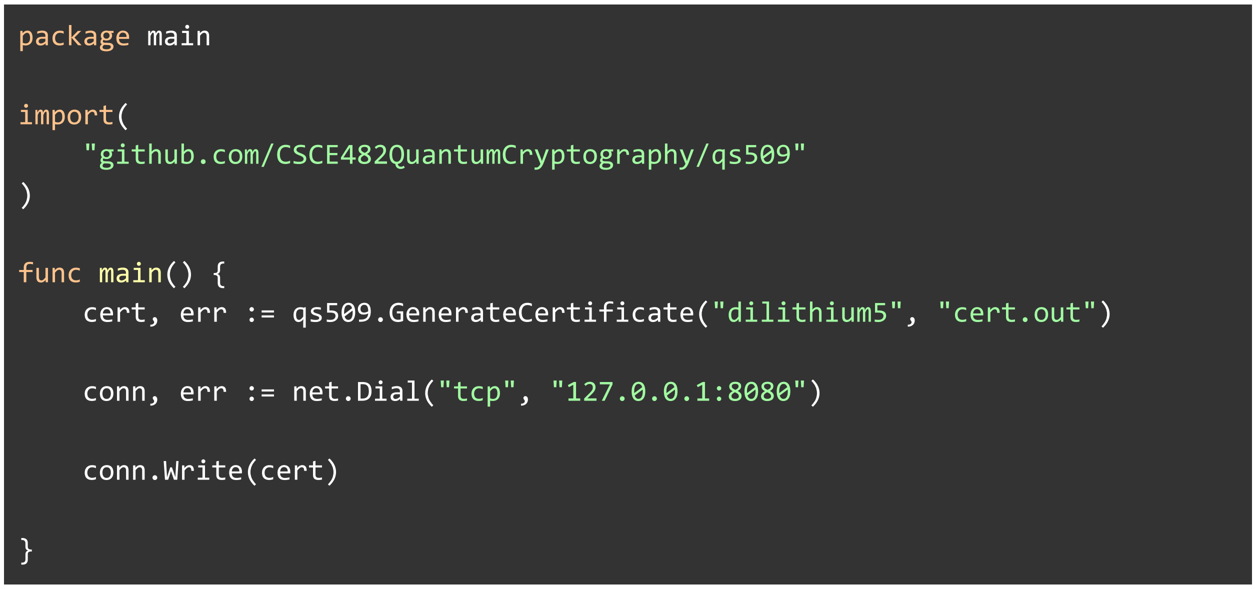
parallels@ubuntu-linux-22-04-desktop: ~/quantumsafe/ProjectCode/server 204x9

```
parallels@ubuntu-linux-22-04-desktop:~/quantumsafe/ProjectCode/server$ go run server.go
Started Listening
Received client public key!
Sending client shared secret in cipher!
Received IV: [130 151 178 19 41 144 70 75 33 217 39 139 90 217 105 143]
Hello 127.0.0.1:51388
Data: Hello server! This is an encrypted message! 43
```

parallels@ubuntu-linux-22-04-desktop: ~/quantumsafe/ProjectCode/client 204x44

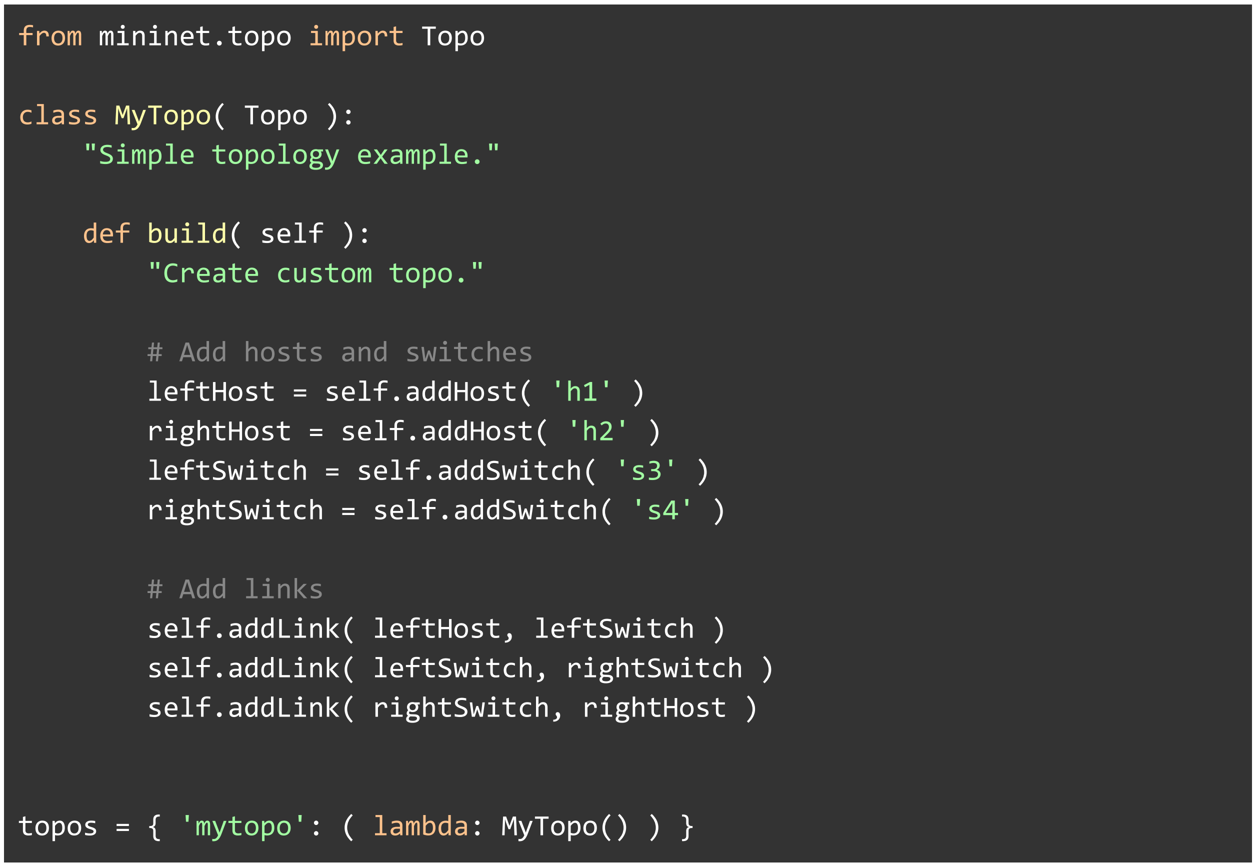
```
parallels@ubuntu-linux-22-04-desktop:~/quantumsafe/ProjectCode/client$ go run client.go

KEM details:
Name: Kyber512
Version: https://github.com/pq-crystals/kyber/commit/74cad307858b61e434490c75f812cb9b9ef7279b
Claimed NIST level: 1
Is IND_CCA: true
Length public key (bytes): 800
Length secret key (bytes): 1632
Length ciphertext (bytes): 768
Length shared secret (bytes): 32

Sending public kyber key to server!
Received shared secret from server!
IV Sent: [130 151 178 19 41 144 70 75 33 217 39 139 90 217 105 143]
Text to send (q to exit): Hello server! This is an encrypted message!
Encrypted Data Written: [101 8 60 40 246 149 156 255 18 104 247 119 93 200 78 63 17 59 202 79 234 94 102 48 143 203 171 20 194 54 161 201 184 178 198 185 208 112 42 1 138 213 182] 43
Text to send (q to exit):
```

Figure 5.A Basic Client Tunnel UI



```go
package main

import(
    "github.com/CSCE482QuantumCryptography/qs509"
)

func main() {
    cert, err := qs509.GenerateCertificate("dilithium5", "cert.out")

    conn, err := net.Dial("tcp", "127.0.0.1:8080")

    conn.Write(cert)

}
```

Figure 5.B Library Call UI



```python
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def build( self ):
        "Create custom topo."

        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's3' )
        rightSwitch = self.addSwitch( 's4' )

        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHost )


topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Figure 5.C Mininet Network Prototype

Annex 6: Design Models

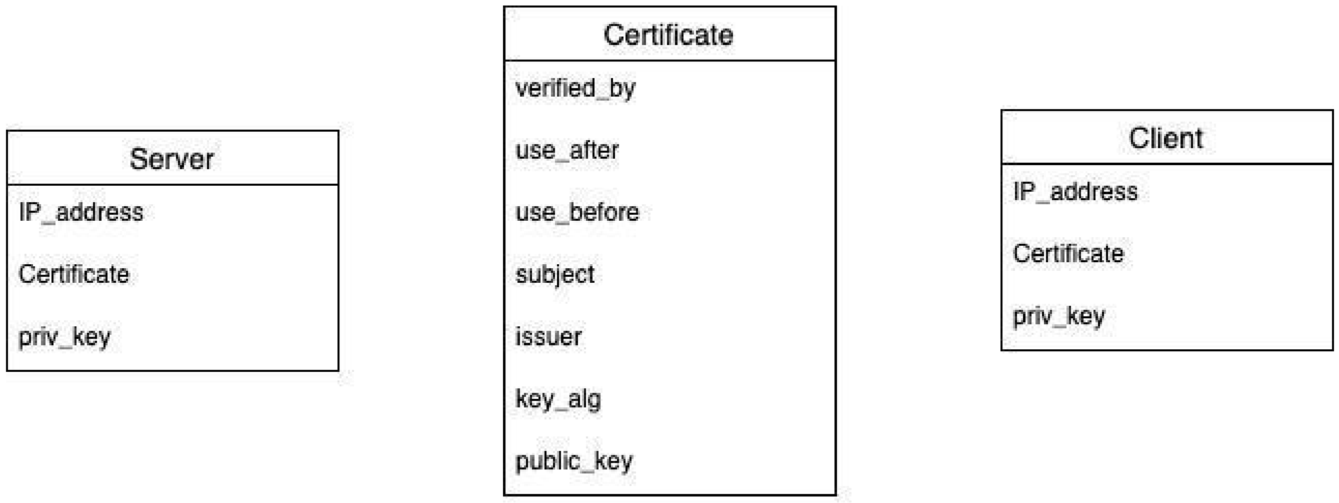


Figure 6.A Data ERD

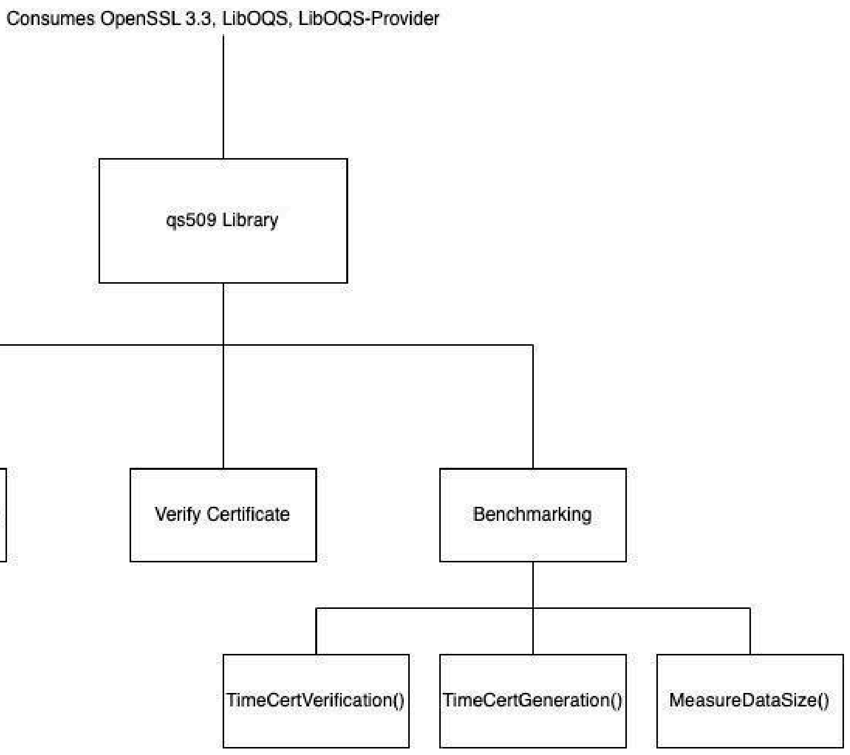


Figure 6.B qs509 Structural Diagram

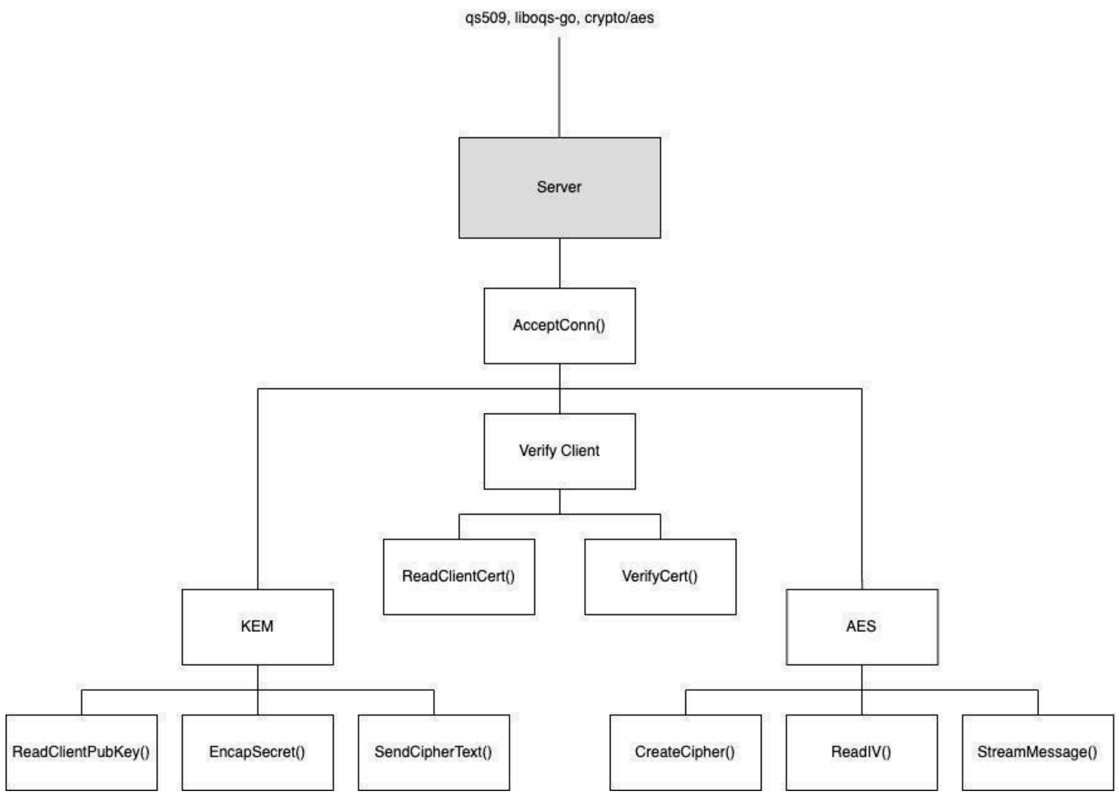


Figure 6.C Server Structural Diagram

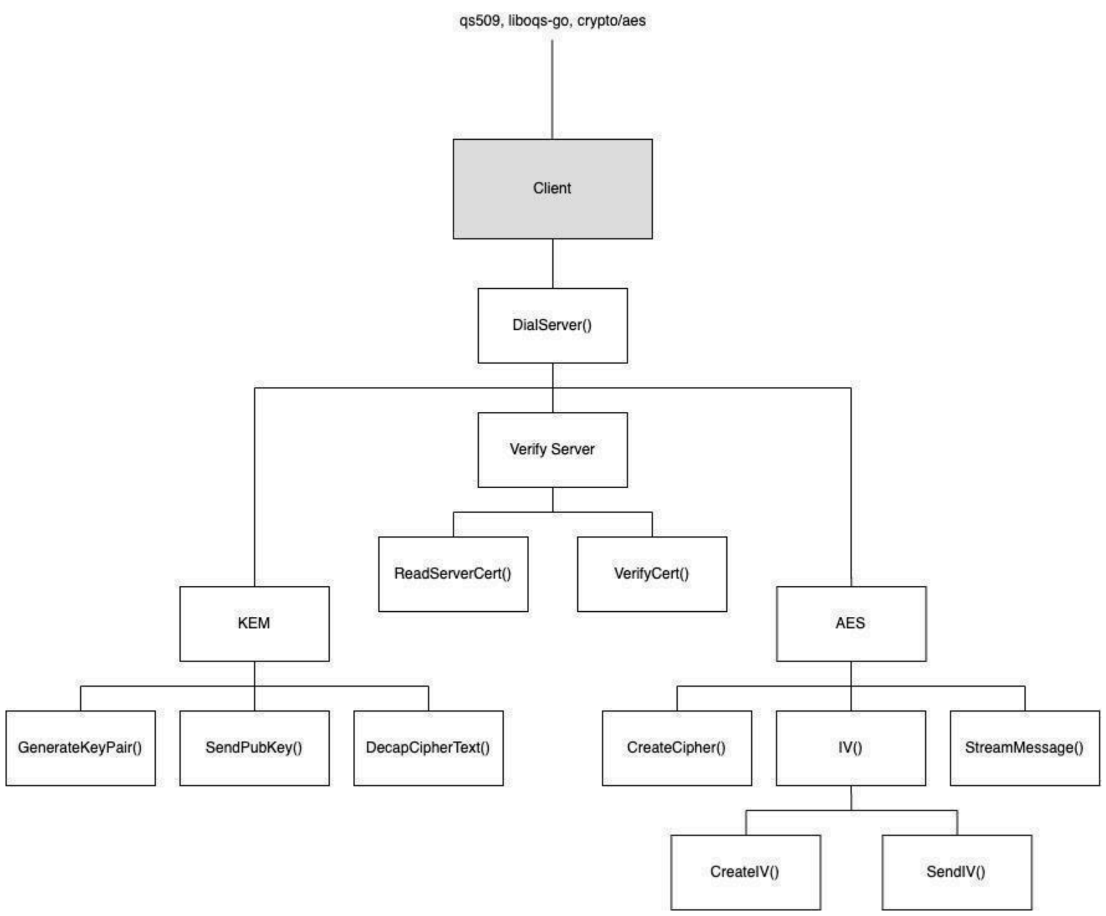


Figure 6.D Client Structural Diagram

Annex 7: Test Cases

Annex 7.1: Unit Tests

Test Case	Input	Expected Output
Sunny Case: Generate Certificate	Generate certificate with valid signing algorithm	A certificate is placed in directory
Rainy Case: Generate Certificate	Generate certificate with invalid signing algorithm	Certificate Generation Fail error
Sunny Case: Generate Key	Generate a key with a valid algorithm specified	Key generation matches the desired one from the user
Rainy Case: Generate Key	Generate a key with an invalid algorithm specified	Generate Key Fail error
Sunny Case: Generate CSR	Request local CA to sign a valid certificate	A csr is placed in director
Rainy Case: Generate CSR	Request CA to sign an invalid certificate	CSR error
Sunny Case: Verify Certificate	Request a valid certificate to be verified	true
Rainy Case: Verify Certificate	Request invalid certificate to be verified	false
Sunny Case: Create / Edit Benchmark Files	Generate an .xlsx file that logs an instance of key and certificate usage. Add to an existing .xlsx file that keeps a running list of all uses.	File created and pre-existing file edited
Rainy Case: Create / Edit Benchmark Files	Fail to generate an .xlsx file that logs an instance of key and certificate usage. Add to an existing .xlsx file that keeps a running list of all uses.	File not created and pre-existing file not edited

Annex 7.2: Integration Tests

Test Case	Input	Expected Output
Sunny Case: Verify Client Cert	Server reads in client cert, calls qs509 to verify with local CA	true
Rainy Case: Verify Client Cert	Server reads in client cert, calls qs509 to verify with unknown CA	false
Rainy Case: Verify Client Cert	Server reads in unsigned client cert, calls qs509 to verify with local CA	false
Sunny Case: Verify Server Cert	Client reads in server cert, calls qs509 to verify with local CA	true
Rainy Case: Verify Server Cert	Client reads in server cert, calls qs509 to verify with unknown CA	false
Rainy Case: Verify Server Cert	Client reads in unsigned server cert, calls qs509 to verify with local CA	false
Sunny Case: Transmit certificate across network	Client converts certificate file to bitstream and streams across network	success
Rainy Case: Transmit certificate across network	Client converts invalid certificate file to bitstream and streams across network	error reading client cert

Annex 8: User Acceptance Test Artifacts

Annex 8.1: Survey Questions

- Please indicate which set of build instructions you have completed
  - Build from Docker
  - Build from Source
  - Watched provided Test Videos
- Were the build instructions sufficient to get you a working environment?
  - Yes
  - No
- Were you able to test the client/server application locally?
  - Yes
  - No
- Rate the UI/UX experience of using our programs. Keep in mind provided documentation, how intuitive the commands are, etc.
  - 1 - 5 (1 is bad experience, 5 is great experience)
- Does the provided data match the expectations? (formatted clearly, provides insightful analysis, etc.)
  - Yes
  - No
- Do we meet the overall minimum agreed upon requirements of the project?
  - Yes
  - No
- What suggestions do you have moving forward in this project?

After each question there will be an opportunity for the user to elaborate on any of the statements with specific examples

Annex 8.2: Survey Results

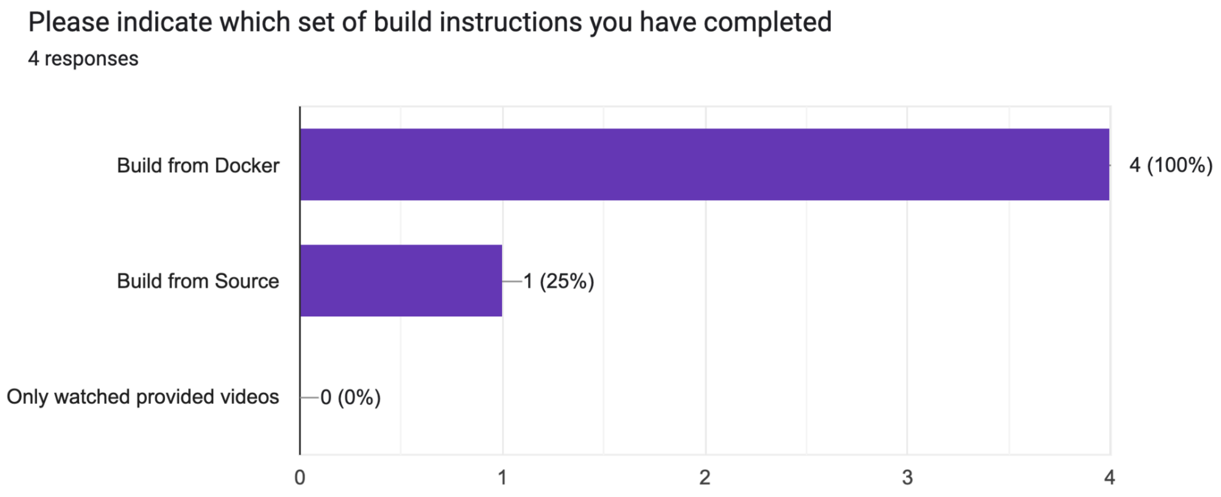


Figure 8.2.A Build Instructions

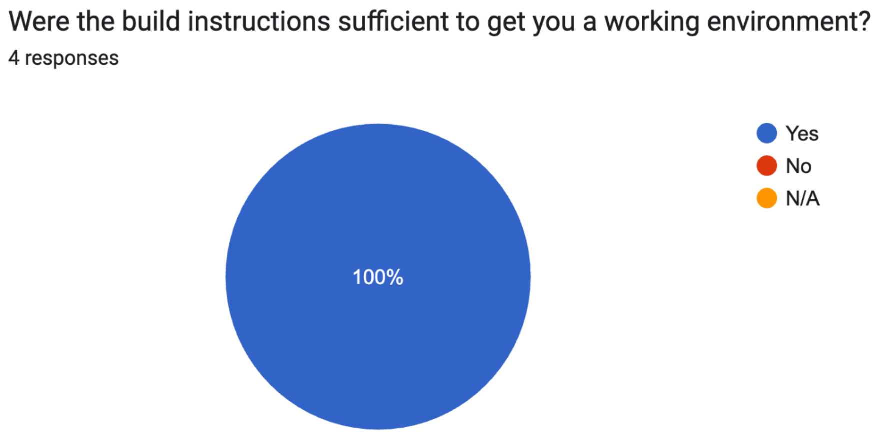


Figure 8.2.B Build Instructions Feedback

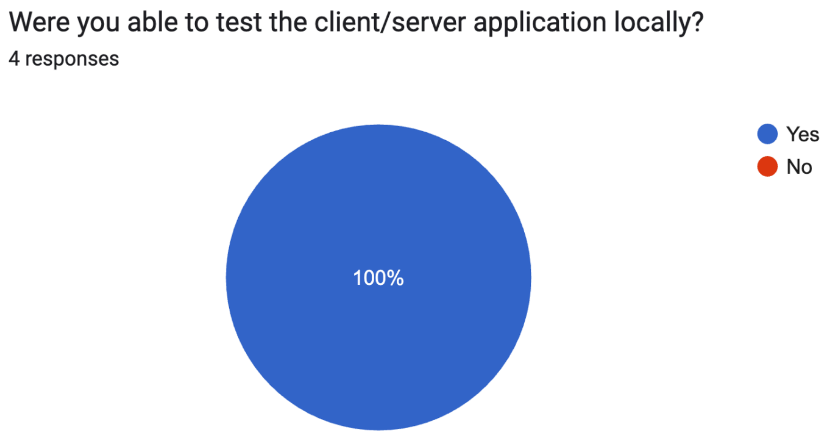


Figure 8.2.C Ability to Test

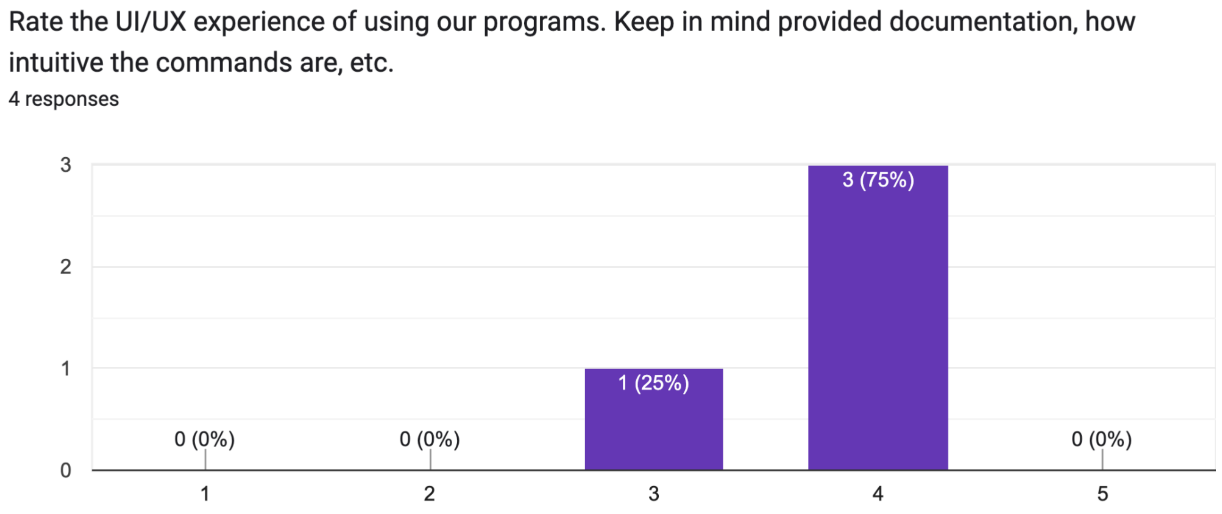


Figure 8.2.D UI/UX Rating

Does the provided data match the expectations? (formatted clearly, provides insightful analysis, etc)

4 responses

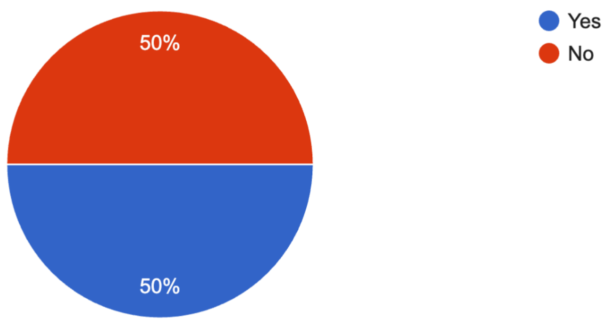


Figure 8.2.E Data Expectations

Do we meet the overall minimum agreed upon requirements of the project? (Our library functionality, our tunnel application, the mininet topologies, the provided timing and wireshark data)

4 responses

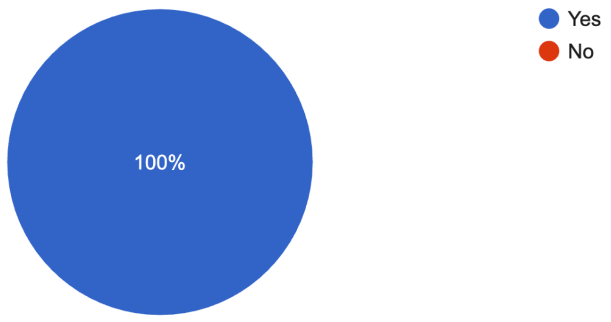
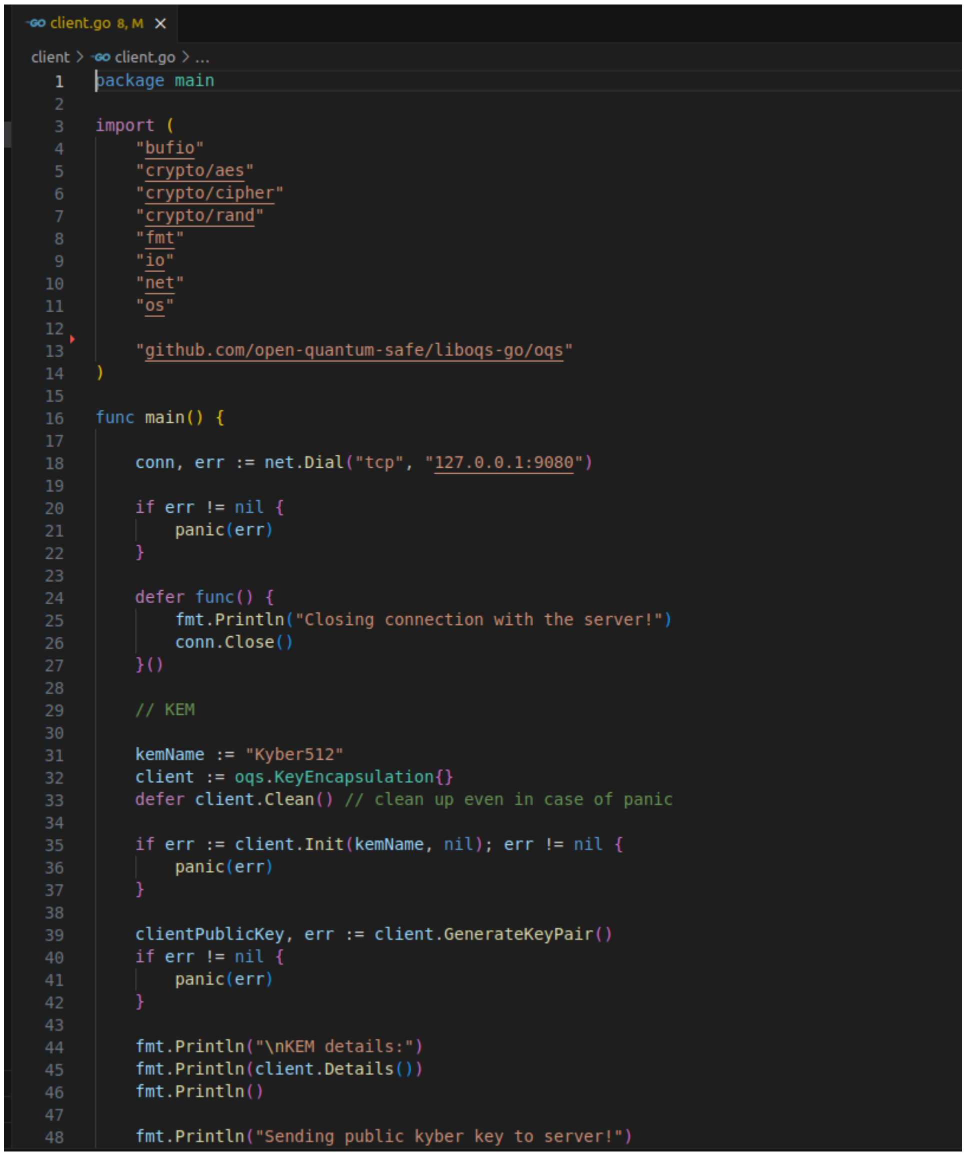


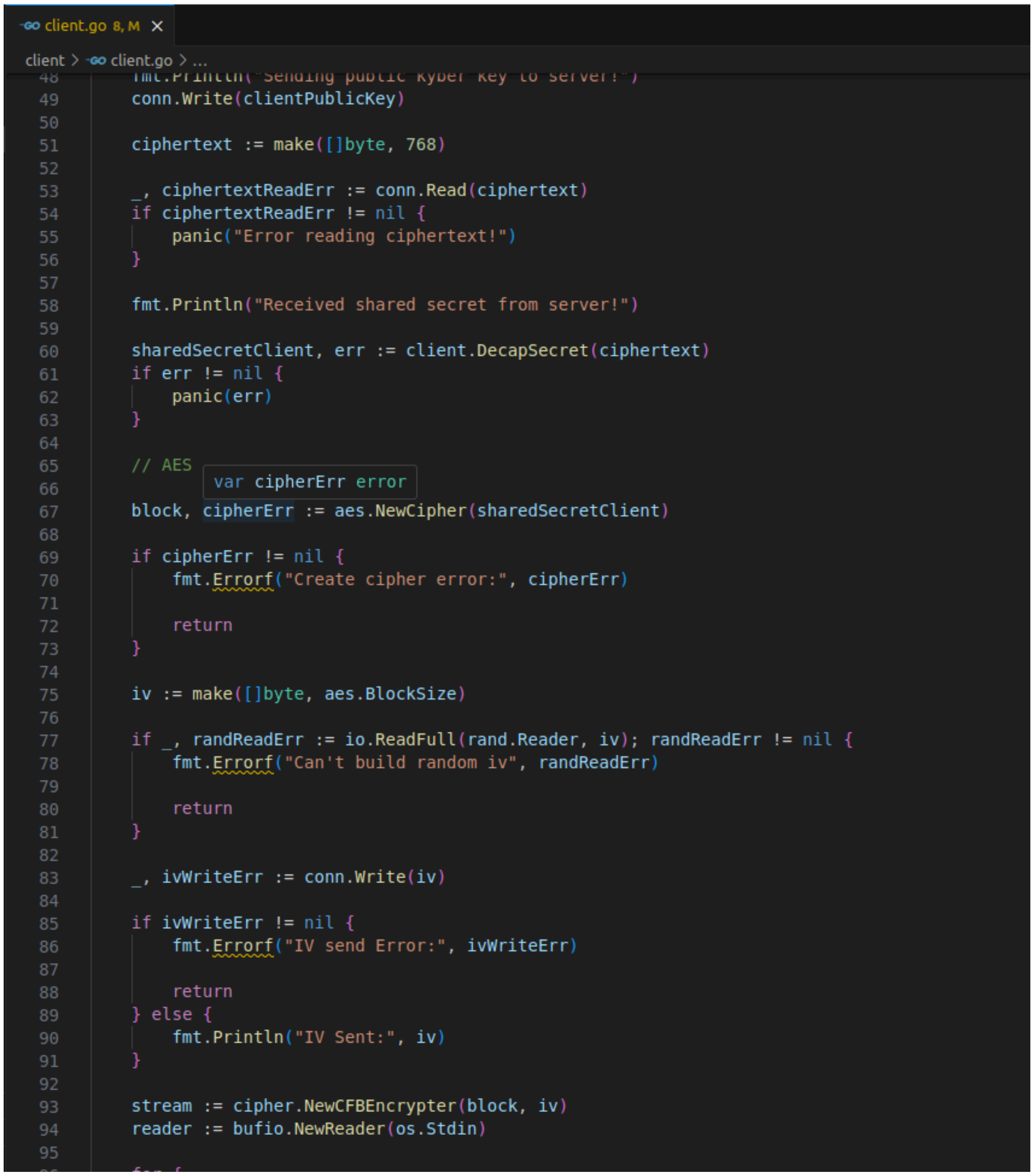
Figure 8.2.F Minimum Requirements Met

Annex 9: Implementation



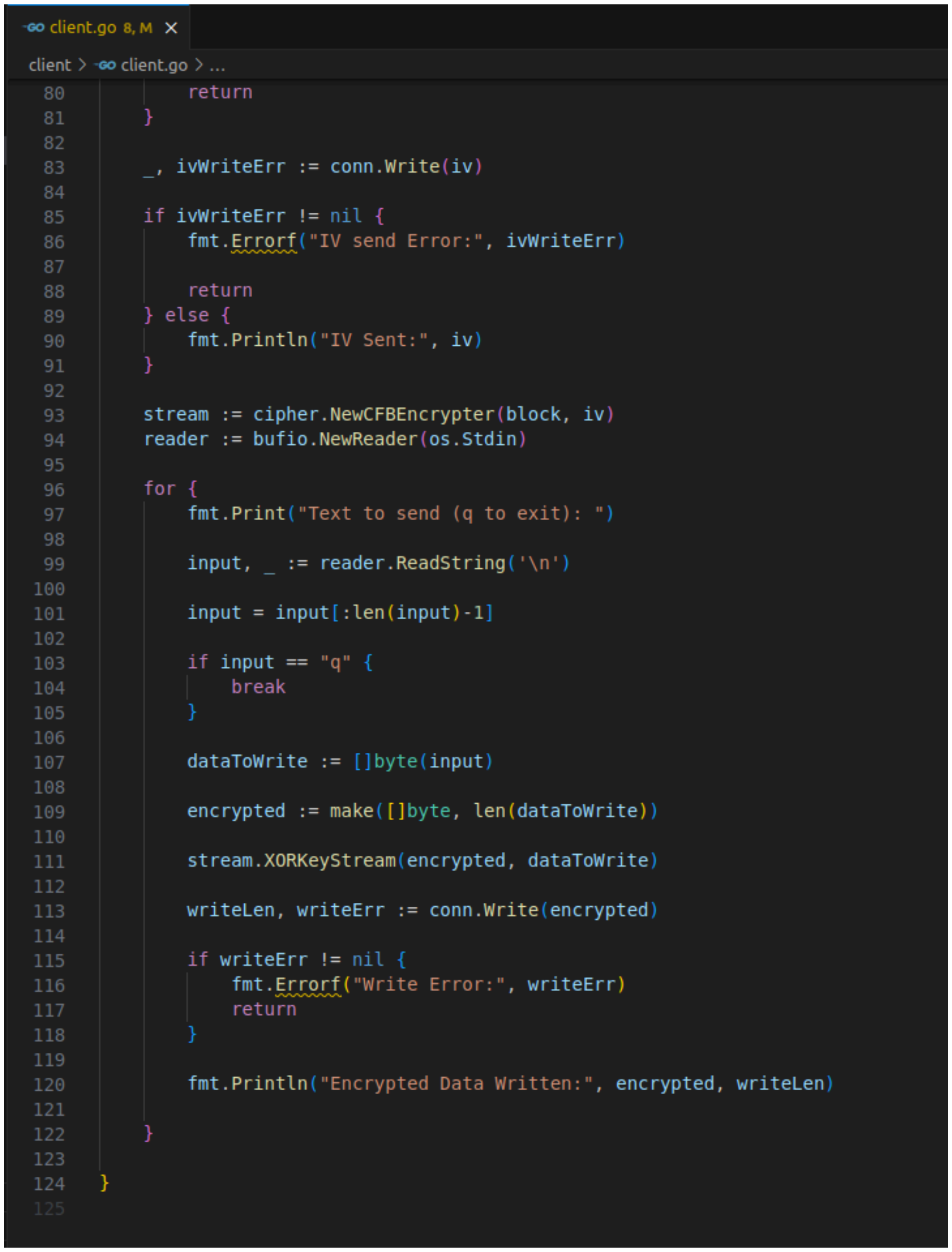
```go
package main

import (
	"bufio"
	"crypto/aes"
	"crypto/cipher"
	"crypto/rand"
	"fmt"
	"io"
	"net"
	"os"

	"github.com/open-quantum-safe/liboqs-go/oqs"
)

func main() {

	conn, err := net.Dial("tcp", "127.0.0.1:9080")

	if err != nil {
		panic(err)
	}

	defer func() {
		fmt.Println("Closing connection with the server!")
		conn.Close()
	}()

	// KEM

	kemName := "Kyber512"
	client := oqs.KeyEncapsulation{}
	defer client.Clean() // clean up even in case of panic

	if err := client.Init(kemName, nil); err != nil {
		panic(err)
	}

	clientPublicKey, err := client.GenerateKeyPair()
	if err != nil {
		panic(err)
	}

	fmt.Println("\nKEM details:")
	fmt.Println(client.Details())
	fmt.Println()

	fmt.Println("Sending public kyber key to server!")
```

Figure 9.A Client Code Part 1



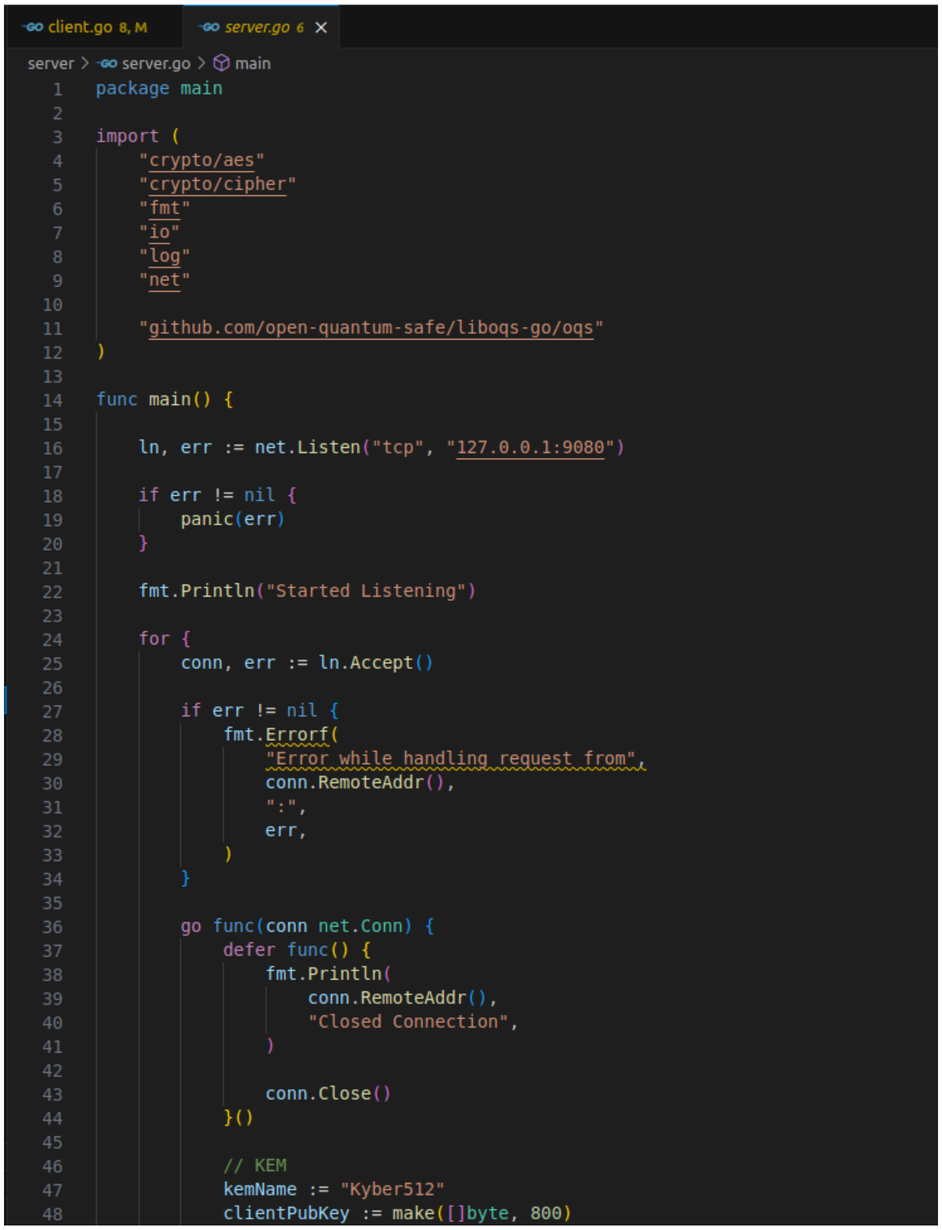
```go
    fmt.Println("Sending public kyber key to server!")
    conn.Write(clientPublicKey)

    ciphertext := make([]byte, 768)

    _, ciphertextReadErr := conn.Read(ciphertext)
    if ciphertextReadErr != nil {
        panic("Error reading ciphertext!")
    }

    fmt.Println("Received shared secret from server!")

    sharedSecretClient, err := client.DecapSecret(ciphertext)
    if err != nil {
        panic(err)
    }

    // AES

    block, cipherErr := aes.NewCipher(sharedSecretClient)

    if cipherErr != nil {
        fmt.Errorf("Create cipher error:", cipherErr)

        return
    }

    iv := make([]byte, aes.BlockSize)

    if _, randReadErr := io.ReadFull(rand.Reader, iv); randReadErr != nil {
        fmt.Errorf("Can't build random iv", randReadErr)

        return
    }

    _, ivWriteErr := conn.Write(iv)

    if ivWriteErr != nil {
        fmt.Errorf("IV send Error:", ivWriteErr)

        return
    } else {
        fmt.Println("IV Sent:", iv)
    }

    stream := cipher.NewCFBEncrypter(block, iv)
    reader := bufio.NewReader(os.Stdin)

    for {
```

Figure 9.B Client Code Part 2



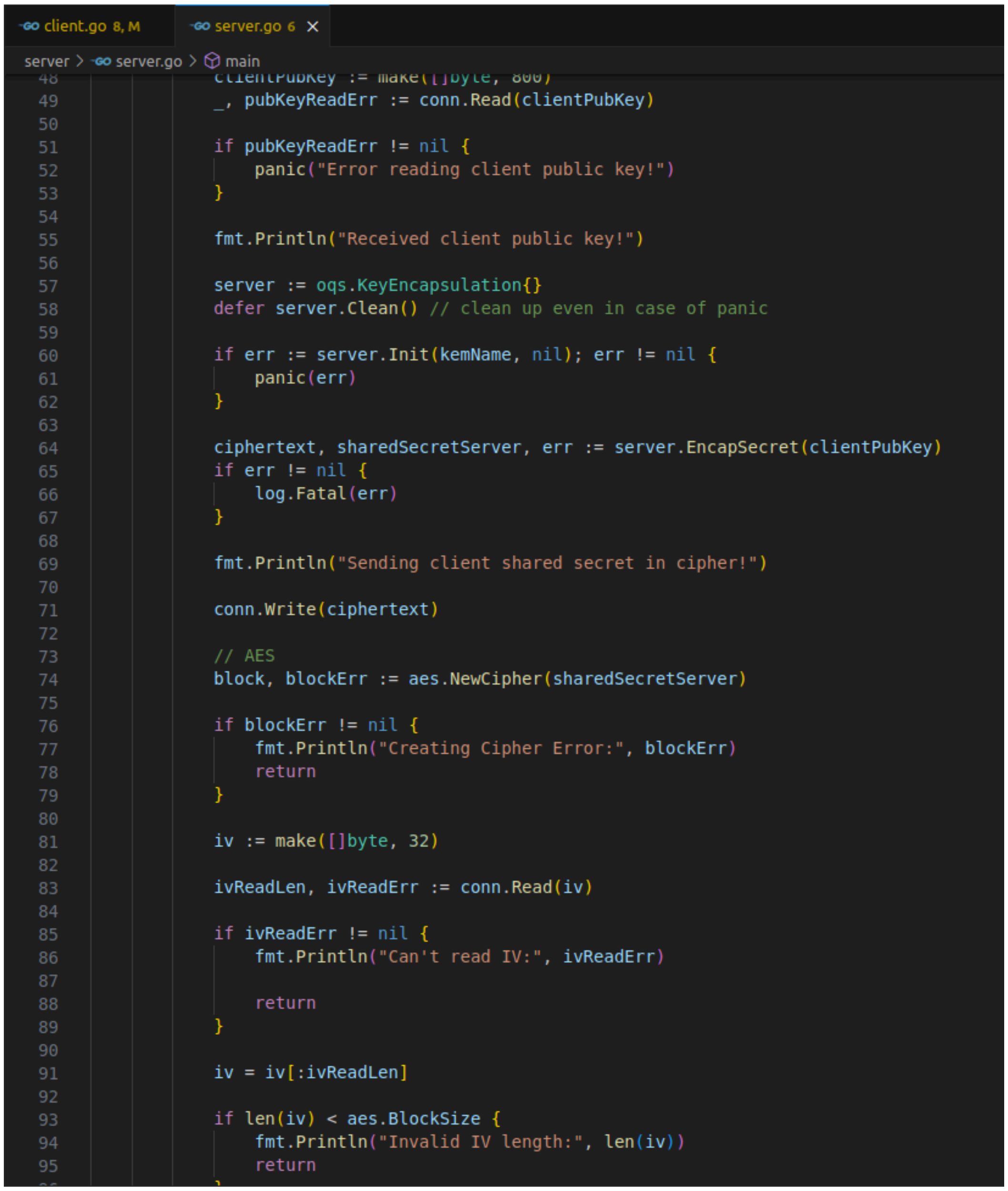
```go
            return
        }

        _, ivWriteErr := conn.Write(iv)

        if ivWriteErr != nil {
            fmt.Errorf("IV send Error:", ivWriteErr)

            return
        } else {
            fmt.Println("IV Sent:", iv)
        }

        stream := cipher.NewCFBEncrypter(block, iv)
        reader := bufio.NewReader(os.Stdin)

        for {
            fmt.Print("Text to send (q to exit): ")

            input, _ := reader.ReadString('\n')

            input = input[:len(input)-1]

            if input == "q" {
                break
            }

            dataToWrite := []byte(input)

            encrypted := make([]byte, len(dataToWrite))

            stream.XORKeyStream(encrypted, dataToWrite)

            writeLen, writeErr := conn.Write(encrypted)

            if writeErr != nil {
                fmt.Errorf("Write Error:", writeErr)
                return
            }

            fmt.Println("Encrypted Data Written:", encrypted, writeLen)

        }

    }
```

Figure 9.C Client Code Part 3



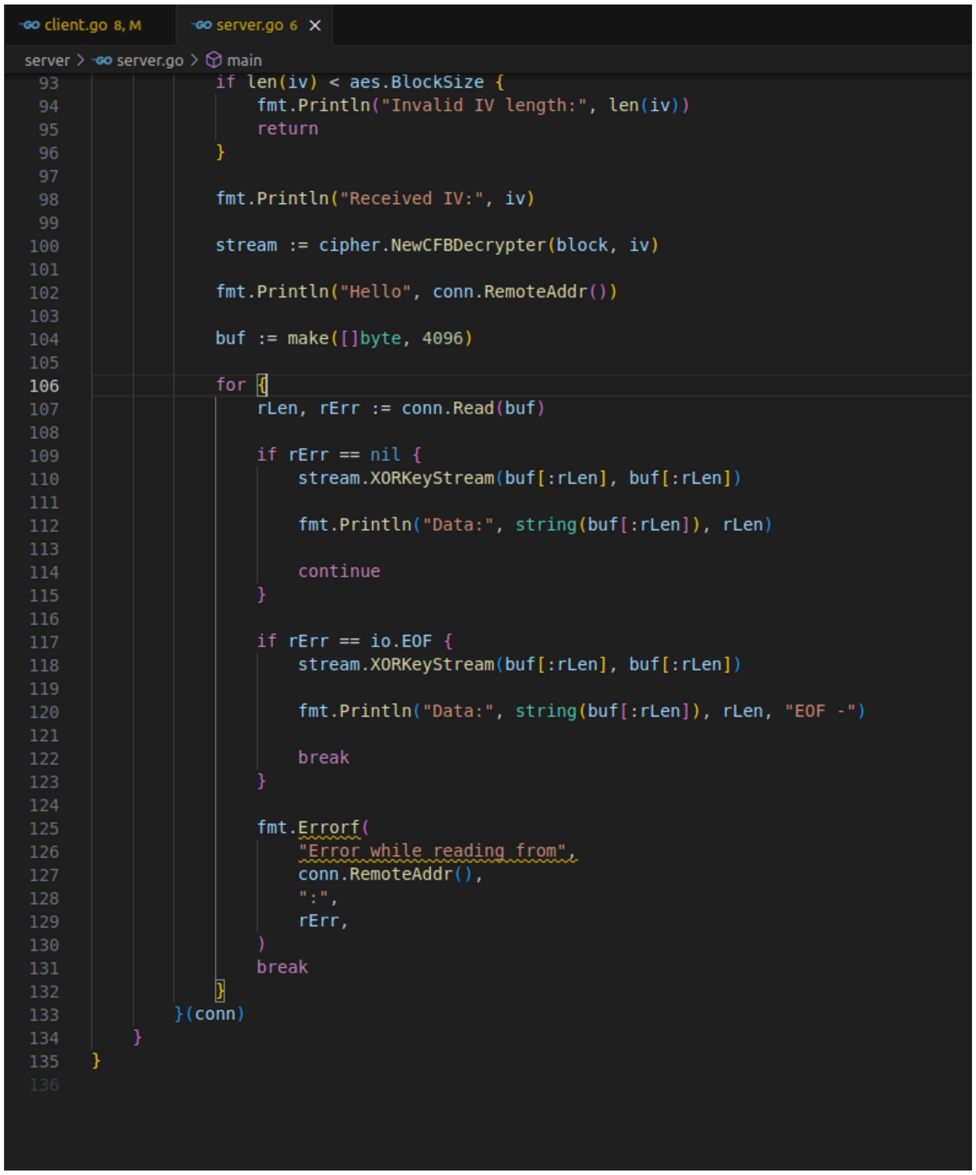
```go
package main

import (
	"crypto/aes"
	"crypto/cipher"
	"fmt"
	"io"
	"log"
	"net"

	"github.com/open-quantum-safe/liboqs-go/oqs"
)

func main() {

	ln, err := net.Listen("tcp", "127.0.0.1:9080")

	if err != nil {
		panic(err)
	}

	fmt.Println("Started Listening")

	for {
		conn, err := ln.Accept()

		if err != nil {
			fmt.Errorf(
				"Error while handling request from",
				conn.RemoteAddr(),
				":",
				err,
			)
		}

		go func(conn net.Conn) {
			defer func() {
				fmt.Println(
					conn.RemoteAddr(),
					"Closed Connection",
				)

				conn.Close()
			}()

			// KEM
			kemName := "Kyber512"
			clientPubKey := make([]byte, 800)
```

Figure 9.D Server Code Part 1



```go
		clientPubKey := make([]byte, 800)
		_, pubKeyReadErr := conn.Read(clientPubKey)

		if pubKeyReadErr != nil {
			panic("Error reading client public key!")
		}

		fmt.Println("Received client public key!")

		server := oqs.KeyEncapsulation{}
		defer server.Clean() // clean up even in case of panic

		if err := server.Init(kemName, nil); err != nil {
			panic(err)
		}

		ciphertext, sharedSecretServer, err := server.EncapSecret(clientPubKey)
		if err != nil {
			log.Fatal(err)
		}

		fmt.Println("Sending client shared secret in cipher!")

		conn.Write(ciphertext)

		// AES
		block, blockErr := aes.NewCipher(sharedSecretServer)

		if blockErr != nil {
			fmt.Println("Creating Cipher Error:", blockErr)
			return
		}

		iv := make([]byte, 32)

		ivReadLen, ivReadErr := conn.Read(iv)

		if ivReadErr != nil {
			fmt.Println("Can't read IV:", ivReadErr)

			return
		}

		iv = iv[:ivReadLen]

		if len(iv) < aes.BlockSize {
			fmt.Println("Invalid IV length:", len(iv))
			return
```

Figure 9.E Server Code Part 2



```go
                if len(iv) < aes.BlockSize {
                    fmt.Println("Invalid IV length:", len(iv))
                    return
                }

                fmt.Println("Received IV:", iv)

                stream := cipher.NewCFBDecrypter(block, iv)

                fmt.Println("Hello", conn.RemoteAddr())

                buf := make([]byte, 4096)

                for {
                    rLen, rErr := conn.Read(buf)

                    if rErr == nil {
                        stream.XORKeyStream(buf[:rLen], buf[:rLen])

                        fmt.Println("Data:", string(buf[:rLen]), rLen)

                        continue
                    }

                    if rErr == io.EOF {
                        stream.XORKeyStream(buf[:rLen], buf[:rLen])

                        fmt.Println("Data:", string(buf[:rLen]), rLen, "EOF -")

                        break
                    }

                    fmt.Errorf(
                        "Error while reading from",
                        conn.RemoteAddr(),
                        ":",
                        rErr,
                    )
                    break
                }
            }(conn)
        }
    }
```

Figure 9.F Server Code Part 3

Annex 10: User's Manual

There are three separate manuals outlined below for setting up the different portions of the project. Starting off, we need to make sure we are using the correct machine and running the correct versions of Go. We will look into building and using the qs509 Go Library we have developed. Afterwards, build instructions will be provided for the client/server application.

Along with the manual provided here, we also provide Docker containers and instructions on how to build and deploy these containers. These tutorials can be found here.

Annex 10.1: Environment

Our qs509 library and our client/server application requires the use of a couple specific softwares. Firstly, we need to be running in an Ubuntu environment. This helps make everything standard and provides us with helpful build tools.

- Any Ubuntu version 20.04 or above will work

Next, we require you to install Go, as everything is compiled and run with Go commands.

- Any Go version 1.20 or above will work

To set up these environment requirements, simply follow the set up instructions on your virtual machine software and the go development website.

Annex 10.2: qs509 Library

Our qs509 library is dependent on three other packages, namely:

- LibOQS
- OpenSSL 3.3
- OQS-provider

To install these three packages, follow the following steps:

Annex 10.2.1: Installing Dependencies

Starting from the root of your terminal, run the following commands to create a workspace for all quantum-safe packages:

```
# If you are not running as root you might need to use "sudo apt" instead
sudo apt update
sudo apt -y install git build-essential perl cmake autoconf libtool zlib1g-dev

export WORKSPACE=~/quantumsafe # set this to a working dir of your choice
export BUILD_DIR=$WORKSPACE/build # this will contain all the build artifacts
```

```
mkdir -p $BUILD_DIR/lib64
ln -s $BUILD_DIR/lib64 $BUILD_DIR/lib
```

Annex 10.2.2: Installing OpenSSL 3.3

```
cd $WORKSPACE

git clone https://github.com/openssl/openssl.git
cd openssl

./Configure \
  --prefix=$BUILD_DIR \
  no-ssl no-tls1 no-tls1_1 no-afalgeng \
  no-shared threads -lm

make -j $(nproc)
make -j $(nproc) install_sw install_ssldirs
```

Annex 10.2.3: Installing LibOQS

```
cd $WORKSPACE

git clone https://github.com/open-quantum-safe/liboqs.git
cd liboqs

mkdir build && cd build

cmake \
  -DCMAKE_INSTALL_PREFIX=$BUILD_DIR \
  -DBUILD_SHARED_LIBS=ON \
  -DOQS_USE_OPENSSL=OFF \
  -DCMAKE_BUILD_TYPE=Release \
  -DOQS_BUILD_ONLY_LIB=ON \
  -DOQS_DIST_BUILD=ON \
  ..

make -j $(nproc)
make -j $(nproc) install
```

Annex 10.2.4: Installing OQS-provider

```
cd $WORKSPACE

git clone https://github.com/open-quantum-safe/oqs-provider.git
cd oqs-provider

liboqs_DIR=$BUILD_DIR cmake \
  -DCMAKE_INSTALL_PREFIX=$WORKSPACE/oqs-provider \
  -DOPENSSL_ROOT_DIR=$BUILD_DIR \
  -DCMAKE_BUILD_TYPE=Release \
  -S . \
  -B _build
cmake --build _build

# Manually copy the lib files into the build dir
cp _build/lib/* $BUILD_DIR/lib/

# We need to edit the openssl config to use the oqsprovider
sed -i "s/default = default_sect/default = default_sect\noqsprovider = oqsprovider_sect/g"
$BUILD_DIR/ssl/openssl.cnf &&
sed -i "s/\[default_sect\]/\[default_sect\]\nactivate = 1\n\[oqsprovider_sect\]\nactivate =
1\n/g" $BUILD_DIR/ssl/openssl.cnf

# These env vars need to be set for the oqsprovider to be used when using OpenSSL
export OPENSSL_CONF=$BUILD_DIR/ssl/openssl.cnf
export OPENSSL_MODULES=$BUILD_DIR/lib
$BUILD_DIR/bin/openssl list -providers -verbose -provider oqsprovider
```

At this point, there should be a fully configured directory at the root of your machine named “quantumsafe”. Going forwards, everything should be happening within this directory.

Annex 10.2.5: Cloning qs509

```
cd $WORKSPACE

git clone https://github.com/CSCE482QuantumCryptography/qs509.git
```

And now at this point, you have fully configured the qs509 library from the source. You can test individual functions by running the following command:

```
go test -v -run <Name_of_Function_Test>
```

Annex 10.3: Client/Server Application

Annex 10.3.1: Liboqs-Go

This application relies on one more package that we have not yet configured, this is the liboqs-go library, which provides functionality for the KEM process in the tunnel. To configure this, follow the steps below:

You need to export a few environment variables, these variables will all point to portions of the liboqs directory built during the setup of the qs509 library. The liboqs directory can be found in your quantum safe workspace.

```
cd $WORKSPACE
```

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$WORKSPACE/liboqs/build/lib
```

Now, we need to install liboqs-go.

```
git clone --depth=1 https://github.com/open-quantum-safe/liboqs-go
```

This provides us with a new package in our quantumsafe workspace named “liboqs-go”. We need to tell this package where it can find the liboqs, so open up the liboqs-go configuration directory in a code editor:

```
cd $WORKSPACE/liboqs-go/.config
code .
```

In the file “liboqs-go.pc”, change the very first two lines to:

```
LIBOQS_INCLUDE_DIR=$WORKSPACE/liboqs/build/include
LIBOQS_LIB_DIR=$WORKSPACE/liboqs/build/lib
```

Save the file and exit.

Finally, all we need to do now is export an environment variable to this updated configuration file. This requires you have pkg-config installed on your machine, so if you do not run this install command first:

```
sudo apt-get install pkg-config
```

Then run:

```
export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$WORKSPACE/liboqs-go/.config
```

Annex 10.3.2: Client/Server Source Code

```
cd $WORKSPACE

git clone https://github.com/CSCE482QuantumCryptography/client.git
git clone https://github.com/CSCE482QuantumCryptography/server.git
```

And viola! you have installed everything you need for the client/server application.

All that is left at this point is to run the client and server!

```
cd $WORKSPACE/server

go run server.go
```

```
cd $WORKSPACE/client

go run client.go
```