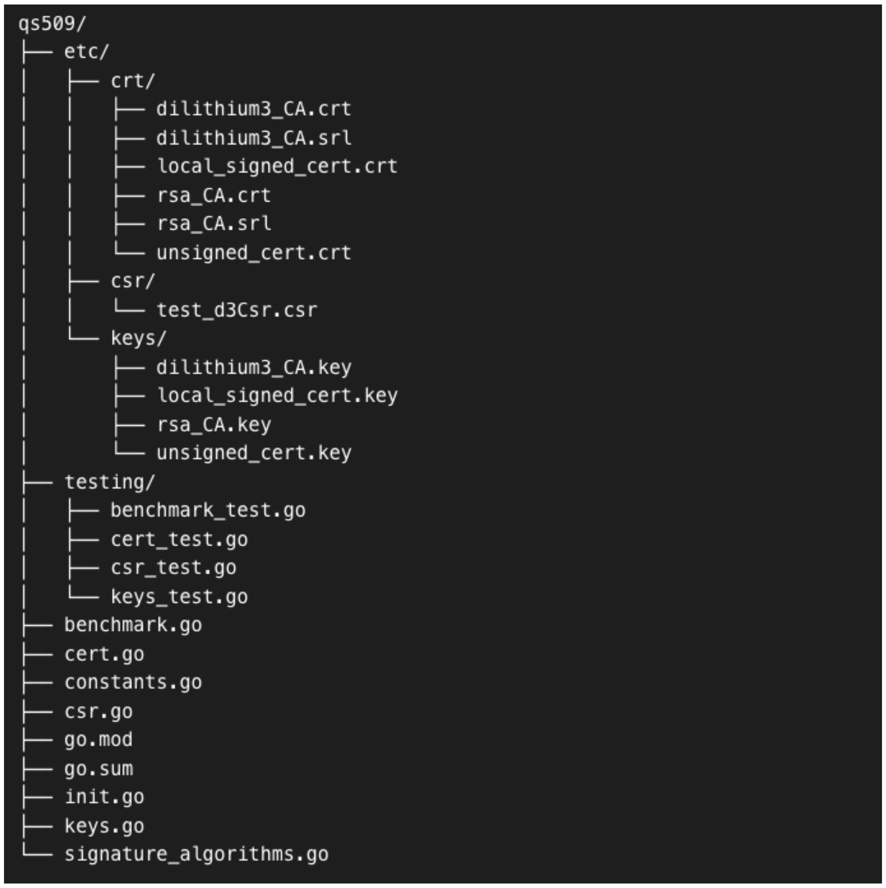
Overview

The purpose of this document is to outline the structure and functionality of our working project. It will dive into each section individually (qs509 library, client and server), elaborating on the different functions, file structure, and purpose.

qs509 Library

The purpose of this library is to provide functionality for quantum-safe x509 certificates. This means generating, signing, and verifying these certificates. Along with this, the library also provides functionality for generating post-quantum keys should an application want to use our library for this reason.

This library is laid out as such:



```
qs509/
├── etc/
│   ├── crt/
│   │   ├── dilithium3_CA.crt
│   │   ├── dilithium3_CA.srl
│   │   ├── local_signed_cert.crt
│   │   ├── rsa_CA.crt
│   │   ├── rsa_CA.srl
│   │   └── unsigned_cert.crt
│   ├── csr/
│   │   └── test_d3Csr.csr
│   └── keys/
│       ├── dilithium3_CA.key
│       ├── local_signed_cert.key
│       ├── rsa_CA.key
│       └── unsigned_cert.key
├── testing/
│   ├── benchmark_test.go
│   ├── cert_test.go
│   ├── csr_test.go
│   └── keys_test.go
├── benchmark.go
├── cert.go
├── constants.go
├── csr.go
├── go.mod
├── go.sum
├── init.go
├── keys.go
└── signature_algorithms.go
```

Throughout this library, there are a number of functions implemented and a number of test cases to test the legitimacy of these functions. Following are the functions and their intended and tested purpose:

Benchmark.go

Benchmark(startTime, endTime)
This function takes in two times and finds the total time between them. It then takes this total execution time and places it within an excel file.

CreateFile(filename)
This function creates an excel file with the specified file name. The file will have two sheets, one named “client”, the other named “server”.

BenchmarkMap(timeMap, sa, ka, outFile, sheet)
This function takes in a map of times saved in an array. For each key in the map, it adds a row in an excel file with the value saved at that key. It then saves this all to the excel file with the specified name. Sa and ka are the signing algorithm and the kem algorithm used during the benchmark.

Cert.go

GenerateCertificate(keyAlg, keyOut, certOut)
This function takes in a specific signing algorithm and then generates a certificate that was created with this algorithm. The certificate is saved to certOut and the corresponding private key is saved to keyOut

VerifyCertificateFile(caCrtPath, certToVerify)
This function takes in a specific certificate file location and a CA certificate file location that is being used to verify the certificate. It returns true if the certificate was signed by the provided CA.

VerifyCertificate(caCrtPath, certBytes)
This function takes in a specific CA certificate file location and a certificate in a byte array. It returns true if the certificate was signed by the provided CA.

Csr.go

GenerateCsr(keyAlg, keyOut, csrOut)
This function takes in a specific signing algorithm and then generates a certificate signing request with this algorithm. The csr is saved to csrOut and the corresponding private key is saved to keyOut

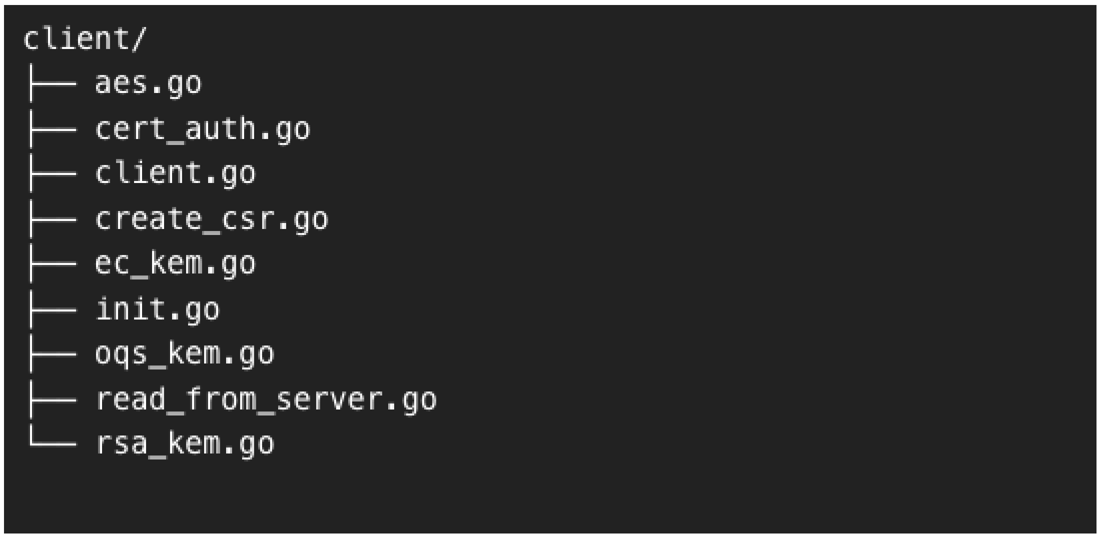
SignCsr(csrPath, crtOut, caCrtPath, caKeyPath)
This function takes a csr and signs it with the CA specified in caCrtPath and caKeyPath. (Our library provides a CA in etc/crt/dilithium3_CA.crt and etc/keys/dilithium3_CA.key). It then returns the signed csr to crtOut.

Keys.go

GenerateKey(keyAlg, keyOut)
This function takes in a key algorithm and generates a key using that algorithm to keyOut.

Client/Server Applications

The purpose of these applications is to set up a local network, go through the process of certificate authentication, KEM, and AES message encryption. These applications act as a way to test the integration of our library with a network system. Both the applications have the following structure (for server, replace client.go with server.go):



```
client/
├── aes.go
├── cert_auth.go
├── client.go
├── create_csr.go
├── ec_kem.go
├── init.go
├── oqs_kem.go
├── read_from_server.go
└── rsa_kem.go
```

The main file in the applications is client.go and server.go. All of the other files simply provide helper functions that are used by the main application (client.go or server.go).

There are a number of different functions used in this library to ensure a safe and secure system, they are explained below.

Client.go and Server.go

This is the main function of the program. It handles measuring the start and end time of all the different functions, creating the connection between server and client

Aes.go

SetupAES(conn, sharedSecret)
This function takes in the connection and the sharedSecret and uses it to set up a new stream that encrypts or decrypts messages sent over the network.

Cert_auth.go

CertAuth(conn, certLen, certFile)
This function takes in the connection and the certificate length and file of whatever application is using it. For clients, the function waits for and verifies the server's certificate. If valid, the client then sends over its certificate. For servers, when a client dials the server, the server immediately sends over its certificate and then waits for the client to verify.

Create_csr.go

CreateCsr()
This function uses the global signing algorithm variable to create a csr and sign that csr with the qs509 library.

Ec_kem.go

ECKem(conn)
This function takes in the connection and then goes through the process of KEM using an Elliptic Curve (256) Diffie-Hellman protocol. The function generates a local key pair, sends the public key to the opposite program, and then derives a shared secret key.

Init.go

init()
This function sets all the global variables used throughout the program as specified by the provided flags.

Oqs_kem.go

OqsKem(conn)
This function goes through the process of KEM with any of the post-quantum KEM algorithms. A client will generate a local key pair, send the public key to the server, and then the server handles encapsulation and creates a cipher for the shared secret. The server finally sends the cipher back to the client who decrypts it with their local private key.

Read_from_server.go and Read_from_client.go

ReadFromServer(conn, buf, readLen) or ReadFromClient(conn, buf, readLen)
These functions repeatedly read packets sent by the opposite application until the buffer is filled with the estimated amount of bytes.

Rsa_kem.go

RSAKem(conn)
This function goes through the process of KEM using RSA. A client will generate a local key pair, send the public key to the server, and then the server handles encapsulation and creates a cipher for the shared secret. The server finally sends the cipher back to the client who decrypts it with their local private key.