COMPX304-22A – Quantum Key Exchange

Ethan McKee-Harris – ID: 1536943

System Design

As a whole, I decided to implement my solution to this assignment in Python as it is my language and lets us do some cool things which will become apparent later.

In terms of inter-item connections, I have decided to implement them by simply calling methods rather than creating socket connections between objects. However, this is just an implementation detail and a socket-based design would also work fine with minimal changes.

I decided to implement this assignment in an object-oriented fashion, this allows us to group relevant code together and keeps the entire project organized as it grew.
Within the assignment we have a total of 5 classes:

1. Client
   This class is responsible for receiving and decrypting messages from a 'Server' and facilitating reads against received messages.
2. Server
   This class is responsible for sending encrypted messages to the 'Client' it has established a connection with.
3. Qubit
   The bread and butter for quantum computing, this class encapsulates the behaviour of a given Qubit in accordance with the assignment spec.
4. XOR
   A string based XOR implementation which just simplifies things like key state management throughout the system when compared to a functional style.
5. MITM
   The class which implements the required features for a successful "man in the middle" attack. This utilises a method more commonly known as SSL Stripping where in this class sits in the middle of the Client and the Server rather than eavesdropping the network stream itself.
   This facilitates reads on intercepted messages while forwarding them to the relevant Client/Server.

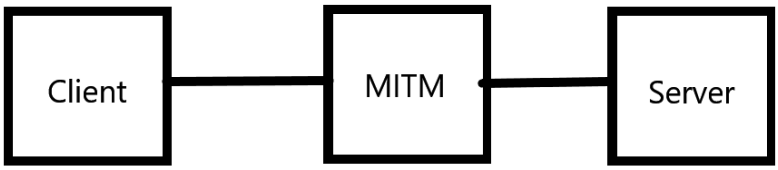
Also, there were various utility features to take this assignment from barebones and turn it into a more fleshed out product. Some of these features included the ability to send messages in either binary or as strings, with the same option existing on the client side in terms of reading.

Testing

All of the implementation was unit-tested with Pytest to 100% coverage. These tests covered both method usage, as well as general coverage of systems working together. Further to this whenever a connection between a Client and Server was established, that test was ran 3 times on the various stream lengths in order to automate key length-based test differentials. You can find these tests as well as the coverage report in the attached assignment directories.

Man In The Middle

As previously mentioned, I decided to exploit an attack vector style more commonly referred to as SSL Stripping. This vector, rather than eavesdropping the network traffic itself actually sets itself up as a middleman between the Client and the Server as seen in the image.



One of the benefits to this style of attack vector is the mitigations it provides towards hiding the breach. As an example, when you attempt to read the encrypted stream while eavesdropping you have the chance to flip the polarization for the Qubit which will produce an error for the real Client/Server who are attempting to decode it. When using SSL stripping, this method for modification and detection is removed due to the man in the middle reading valid connections before encrypting the content using the correct key and forwarding it.

A key point to note with this assignment, however, is the way Qubits are represented versus how they can function in the real world. Given two qubits linked using quantum entanglement, you in theory should not be able to conduct a man in the middle style of attack without physical access to at-least one of the qubits themselves, and at that point you already have the qubit, so this attack style is unrequired.

Mitigations

In order to mitigate possible man in the middles for this style of algorithm we need to find a secure way to establish connections, so we don't leak the key while also providing a means to verify the connection comes from the person we assume it is and not someone else.

One approach could be using a form of pre-established public/private key which is used to encrypt the key transfer before communication is switched to the new key, however, this all depends on pre-existing algorithms which solve the issue. This leads to conversations surrounding the need for quantum key exchange given the reliance on technology which already resolves these issues.
A novel example of this approach would be to use Diffie-Hellman to transmit the keys before changing over to the transmitted keys.

Another form of mitigation would be some form of host verification, that they client is who they say they are. For example, we can look at SSL/TLS in terms of certificate verification. This uses a system where-in any connections verify they are who they say they are by using certificates and working the way up until they hit a certificate from a trusted certificate authority. We could use a similar mechanism to mitigate man in the middles within this space.