Programming a Quantum Network
CS4090 - Quantum Communication and Cryptography

Andrew Jiang - Student ID: 4942795 - A.Y.Jiang@student.tudelft.nl
Boris Joukovsky - Student ID: 4931165 - B.J.Joukovsky@student.tudelft.nl
Olivier Maas - Student ID: 4606159 - O.P.Maas@student.tudelft.nl
Antal Szava - Student ID: 4958489 - A.Szava@student.tudelft.nl

January 17, 2019

1 Introduction

The objective of this project is to simulate a quantum network in order to improve our understanding of the content from the course *Quantum Communication and Cryptography*. As most of the course material culminates to the implementation of a Quantum Key Distribution (QKD) protocol, the project aims to implement a simulation of the BB84 protocol between Alice and Bob, with an interceptor Eve. This project assumes the existence of a classical authenticated channel between Alice and Bob, and assumes that the quantum channel will be noise-free. Eve will attempt to intercept and measure the qubits randomly in the standard or Hadamard basis in an attempt to gain some insight. Alice and Bob will then proceed and measure the error rate in an attempt to detect this.

The source code of the project can be found on the following GitHub repository: `https://github.com/bjoukovs/QuantumCryptoBB84`

2 Code Structure

The three peers of the quantum network were implemented in three respective files: `alice.py`, `bob.py` and `eve.py`. Each one of them make use of a `CQCConnection()` object as an interface to the simulated quantum network. Since these three peers also share common methods, a `helper.py` file has also been written which can be accessed by all peers. Finally, each of the peers can also instantiate an `Extractor()` object from `extractor.py` to perform the privacy amplification step.

- `alice.py` specifies Alice's role in the BB84 protocol. Namely, she defines the number of qubits N that will be used for the protocol and send it to Bob. She then picks a random string of basis and measurement outcomes, which determine the BB84 states to be sent to Bob through Eve. Note that the qubits are sent sequentially using an acknowledgement mechanism, allowing to exceed the 32 qubits limit of SimulaQron, but also ensuring security against a man-in-the middle (MitM) attack performed by Eve (more detail may be found on this in Section 3.2). The verification step is then performed as described in the next section.
- `bob.py` specifies Bob's role in the BB84 protocol. Namely, he receives Alice's qubits and acknowledges them. He then perform the outcome measurements comparison and key extraction as described later on.
- `eve.py` specifies Eve's role in the BB84 protocol. Namely, she acts as an adversary and an intermediate peer between Alice and Bob. Her main role is to route the messages from one peer to

the other. In order to do so, messages are tagged using an integer uniquely identifying its sender and destination, such that Eve can easily forward the messages. An attack consisting of measurements of the qubits in the standard or Hadamard basis (picked randomly) can be activated using `intercept=True`

- `helper.py` contains the common methods to Alice, Bob and Eve such as the basis and measurement comparison methods, sending and parsing the tagged messages.
- `extractor.py` performs the key extraction using a random or specified seed, as described in the next section.

3 Exercises

3.1 Alice sends a random BB84 state to Bob, who measures it in a random basis

In `alice.py`, Alice generates a random bit θ corresponding to the `basis` (with 0 and 1 standing for the standard and the Hadamard basis, respectively) as well as a random bit x to create a qubit $|\theta_x\rangle$, where:

$$|\theta_x\rangle = \begin{cases} |0\rangle & \text{if} \quad \theta = 0,\ x = 0 \\ |1\rangle & \text{if} \quad \theta = 0,\ x = 1 \\ |+\rangle & \text{if} \quad \theta = 1,\ x = 0 \\ |-\rangle & \text{if} \quad \theta = 1,\ x = 1 \end{cases} \tag{1}$$

She then sends $|\theta_x\rangle$ to Bob via Eve using the `sendQubit()` method. Bob, in `bob.py`, generates a random basis in the same way, receives the qubit $|\theta_x\rangle$ that Alice sent, and measures it in the basis he had generated.

3.2 Alice sends N such random BB84 states to Bob

After specifying N as the number of qubits in the beginning, Alice announces N to everyone. She then generates N random bases $\bar{\theta} \in \{0,1\}^N$, as well as N random bits 0 or 1 $\bar{x} \in \{0,1\}^N$ determining the set of BB84 states $|\theta_x\rangle^{\otimes N}$.

She then sends these N qubits to Bob via Eve using the aforementioned acknowledgement mechanism. Bob performs the measurement in bases chosen according to a random string that he generated beforehand. Note that Bob must measure the qubit before sending the acknowledgement message to Alice. This is important, as it guarantees that Bob has received the qubits and measured them, and thus prevents a possible MitM attack, whereby Eve stores all the qubits with her quantum memory, and then upon learning all of the bases Alice measured in (Since Alice would send it without acknowledgement from Bob), Eve would gain access to the entire key.

3.3 Alice and Bob extract one bit of key $k \in \{0,1\}$ using a simple extractor and communicating a random seed

Once Alice receives the acknowledgement from Bob that he has measured the qubits, Alice and Bob exchange a random subset of t their basis string $\bar{\theta}_{Alice}$ and $\bar{\theta}_{Bob}$, from which they compare and find the set of bases where $\bar{\theta}_{i_{Alice}} = \bar{\theta}_{i_{Bob}}, \forall i \mid 0 \leq i < N$. From this set, half of the bases are randomly chosen by Alice, and she sends Bob these chosen bases along with her measurement outcomes associated with each basis. Bob then sends back to Alice his measurement outcomes for the same bases. At this point, both Alice and Bob can compare their own measurement outcomes with the other's measurement outcomes

for the specified bases in order to calculate an error rate. Since the channel is assumed to be noise-free, Alice and Bob will abort if the error rate is greater than 0. If there is no error however, Alice and Bob will proceed to extract a key.

The `Extractor` class implements the extractor needed for extracting a key. Using it, in `alice.py` Alice generates a seed and then extracts one bit of key based on the basis string that she has (The exact mathematical procedure is described in the assignment itself). After this, she sends the seed using our custom `sendClassicalMessage()` method to Bob (and to Eve), after which Bob, in `bob.py`, also creates one bit of key using this seed. As mentioned, Eve also receives this seed in `eve.py` as it goes through the classical authenticated channel.

3.4 Eve implements an attack, causing Alice and Bob to measure some error rate

Eve has the opportunity to intercept and measure a qubit that was sent by Alice to Bob. This is implemented in a way, that in `alice.py` Alice uses the `sendQubit()` to send all her qubits first to Eve. If the `intercept` variable is set to `True`, then Eve measures in either the computational or the Hadamard basis (chosen uniformly at random), and finally forwards the (measured) qubit to Bob. This process might alter the qubit that Alice originally sent. For this reason, when Bob receives the qubit and measures in a certain basis he might have a different measurement outcome than Alice. This will result in their error rate not to be equal to 0.

As a comparison, the error rate has been computed for some various choice of key bits lengths n, using $N = 4n$, when Eve performs a measuring attack. The numbers were averaged over 5 experiments:

n ($N = 4n$)	Averaged error rate	Error standard deviation
4	0.443	0.21
8	0.236	0.165
16	0.249	0.106
32	0.223	0.061

Table 1: Error rates averaged over 5 experiments

Results show that the average error rates approaches 25% as expected from this kind of attack. The standard deviation is decreasing with n. The interpretation is that if n is low, the chance of the error rate to be of a more extreme value is higher, due to the lower number of measurement outcomes to be compared, or samples. This is dangerous since, in the low error rate case, it induces a higher chance of mistakenly identifying the protocol as successful despite Eve's attack.

4 Conclusion

The basic BB84 protocol was successfully implemented, allowing for an arbitrary number of qubits to be exchanged and used as a means to produce a secret key between Alice and Bob. Experiments clearly show a non-null error rate when Eve performs a basic measuring attack. The project could be extended to using a more secure random number generator than the random function built into Python by default; it could also use a different kind of extractor to perform the privacy amplification step and to produce more bits of key than 1. The channel could also be extended to include quantum noise and to perform error reconciliation. the protocol could also be extended to using the purified BB84 version, which would be a good first step towards implementing Device-Independent Quantum Key Distribution (DIQKD).