CMPUT 396, Fall 2019, Assignment 10

Before beginning work on this assignment, carefully read the Assignment Submission Specifications posted on eClass.

You will submit two files for this assignment: a modified "alice.py" module and a README file (also in either PDF or plain text). Do not modify or submit "bob.py" **If your code requires any external modules or other files to run, include them in your submission as well. Your submission should be self-contained**

Problem 1

The module included with this assignment, "bob.py" includes several functions to simulate programming with a quantum channel that can transmit photons for use in quantum key exchange. The "problem1(bob, message)" function in "alice.py" contains comments describing the steps to use this system to securely exchange a key with "bob.py" and send a message encrypted using the one-time pad cipher. The "bob.quantum_channel(data)" method transmits a list of photon characters (one of (↖, ↗, ↑, →)) to Bob and Bob will return (through a secure channel of course!) the list of the filters that he used for each character (either x or +). After instructing Bob to dispose of the data he received using an incorrect filter using the "bob.dispose(elements)" method, Alice encrypts her message using the remaining key characters and the one-time pad cipher before finally transmitting the message to Bob. Implement the steps outlined for problem1 in alice.py so that Alice securely exchanges a key with bob and sends the encrypted message using the "bob.message(ciphertext)" method You should study the documentation provided in the methods in "bob.py" for proper usage of each method.

You may test your function by invoking it as follows:

```
from bob import make_bob
problem1(make_bob(problemNumber=1), "HELLO BOB")
```

Problem 2 When Bob reads a photon using the incorrect filter, there is a 50% chance that the orientation will be randomly altered to match the orientation of the filter he used (Eg. if he used + to read ↖ then the result could be ↑ or →) and a 50% that the photon will be blocked completely. Bob does not know in advance how many photons Alice will send him and the number of filters he reports is the number of photons he received. For example, if 5 of the 10 photons Alice sent to Bob were lost then Bob will report the list of the 5 filters he used and None for the photons that were not received. Modify the program provided in "problem1" so that a new function, "problem2", compensates for this error loss. This problem will require that you make successive calls to "bob.quantum_channel(data)" and "bob.dispose(elements)". Bob maintains a key in memory of the sequence of photons he has received. Successive calls to "bob.quantum_channel(data)" will append new photons to the key he has stored in memory and "bob.dispose(elements)" will remove elements from the key has he stored in memory. Note that if Bob uses the correct filter, the photon will not be dropped.

When you are confident that Alice and Bob have the "key" that is at least 5 times the length of the message parameter pass the encrypted message to the "bob.message(string)" method and exit your function

You may test your function by invoking it as follows:

```
from bob import make_bob
problem2(make_bob(problemNumber=2), "HELLO BOB")
```

If done correctly, your program will print: "Bob's message: HELLO BOB" For the purposes of debugging, you may use the "bob.debug()" method to get the current key that Bob has stored.

Problem 3

Up until now, you have assumed that no messages may be intercepted on the line. This may not always be the case Modify the code from problem 1 in the function named problem3 to detect when Eve is eavesdropping and report it by invoking the "bob.report_eve()" method. You can assume that no photons are blocked. You can achieve this by asking Bob to reveal a portion of the key using the "tell" parameter in the "bob.quantum_channel(data, tell)" method. If there are cases where Bob used the correct filter but obtained the wrong results, you know that the line has been intercepted!

Testing your program with

```
from bob import make_bob
problem3(make_bob(problemNumber=3), "HELLO BOB")
```

You should never call "bob.message(ciphertext)" and should instead call "bob.report_eve()" and return every time.

When your program is invoked using the Bob from problem 1 and the line is not intercepted:

```
from bob import make_bob
problem3(make_bob(problemNumber=1), "HELLO BOB")
```

Your program should transmit the encrypted message to Bob using "bob.message(ciphertext)".

Problem 4

Combine your programs from problems 2 and 3 to build a new program in problem3 that can deal with photons being blocked and the line being intercepted.

As before, testing your program with the bob that does not get intercepted (but does block photons)

```
from bob import make_bob
problem4(make_bob(problemNumber=1), "HELLO BOB")
```

Should result in the encrypted message being transmitted to Bob successfully.

When your program is run as follows

```
from bob import make_bob
problem4(make_bob(problemNumber=4), "HELLO BOB")
```

It should report Eve every time and not transmit a message to Bob.