Appendix B

```
# E-91 Simulation
# Author: Evan Anderson
# License: GNU General Public License version 3

import sys
sys.path.append('./lib/qit-code-python')
from qit import *
from qit.lmap import tensor as lmap_tensor
from numpy import *
import numpy as np
import numpy.random as npr

debugOn = True
identityOp = array([[1,0],[0,1]])

# Fancy pants debug function
def debug(content):
  red = "\x1B["
  if debugOn:
    print red + "31;40m" + str(content) + red + "0m"

def makeRotOperator(theta):
  # All measurements made perp to axis or propegation hence phi = pi/2
  return R_nmr(theta, np.pi/2)

# Calculate S value from unused correlation coefficients
def calcBigOlSVal(countMatrix):
  coef = { 'a1b1': 0, 'a1b3': 0, 'a3b1': 0, 'a3b3': 0 }
  for key in countMatrix :
    tempCount = countMatrix[key]
    totCounts = sum(tempCount.values())
    coef[key] = float( tempCount['uu'] + tempCount['dd'] - tempCount['ud'] -
       ↪ tempCount['du'] ) / totCounts

  return coef['a1b1'] - coef['a1b3'] + coef['a3b1'] + coef['a3b3']

def E91_simulate(n=5000, eveDetTol=0.1, eveInterceptRate=1):
  """Runs E91 simulation.

    Keyword arguments:
    n -- number of entangled bits. Default 5000
    eveDetTol -- Detection tolerance to find Eve. default is 10%
  eveInterceptRate -- Eve's interception rate of the stream of particles,
```

```
  ↪ default is 100%
  """

print("\n==========================================================")
print("Simulating the E-91 Protocol for Quantum Key Destibution")
print("========================================================")
print('Using {0} sets of entangled particles\nEve is intercepting {1}% of the
  ↪ particles\nWe are attempting to detect her with a tolerance of {2}%\n'.
  ↪ format(n,eveInterceptRate*100,eveDetTol*100))

analyzerAnglesA = [0, np.pi / 4, np.pi / 2]
analyzerAnglesB = [np.pi / 4, np.pi / 2, 3 * np.pi / 4]
privateKey = []

# Count the number of times we get a uu, ud, du or dd result from
# non parralel axis of measurement
# This will be used in Eve detection
countMatrix = { 'a1b1': {'uu': 0, 'ud': 0, 'du': 0, 'dd': 0},
        'a1b3': {'uu': 0, 'ud': 0, 'du': 0, 'dd': 0},
        'a3b1': {'uu': 0, 'ud': 0, 'du': 0, 'dd': 0},
        'a3b3': {'uu': 0, 'ud': 0, 'du': 0, 'dd': 0},
        }

# Alice and Bob have a random choice of 3 analyzers to measure their spins
analyzerChoiceA = npr.random_integers(0, 2, n)
analyzerChoiceB = npr.random_integers(0, 2, n)
analyzerChoiceE = npr.random_integers(0, 2, n)
eveStat = { 'intercepted' : 0, 'choseRight' : 0 }

# An entangled pair of particles is generated (Bell state 3)
singletState = state('bell3')

print('Generating and distributing entangled particles...\n')
for k in range(n):
  aIndex = analyzerChoiceA[k]
  bIndex = analyzerChoiceB[k]
  eIndex = analyzerChoiceE[k]

  # Housekeeping to check for Eve eventually
  pIndex = ''
  countMatrixIndex = ''
  if aIndex == 0:
    countMatrixIndex = 'a1'
  elif aIndex == 2:
    countMatrixIndex = 'a3'
```

```
if bIndex == 0:
  countMatrixIndex += 'b1'
elif bIndex == 2:
  countMatrixIndex += 'b3'

# Setup theta choices and generate appropriate rotation operators
thetaA = analyzerAnglesA[ aIndex ]
rotationA = makeRotOperator(thetaA)
# Generate I2 tensor Rotation on Alices bits
rotationOpA = kron(rotationA, identityOp)

thetaB = analyzerAnglesB[ bIndex ]
rotationB = makeRotOperator(thetaB)
rotationOpB = kron(identityOp, rotationB)

# For the sake of maximal success by Eve, assume she knows Alice's potential
# analyzer angles. In order to go undetected, Eve must then choose the
   ↪ exactl
# Same theta as Alice
thetaE = analyzerAnglesA[ eIndex ]
# -thetaE here due to the way qit works, we want to "undo" the rotation
# done to the state by Alice
rotationE = makeRotOperator(-thetaE)
rotationOpE = kron(rotationE, identityOp) # Kronecker tensor product
# Number of particles interceped, number of times her choice matched Alices

# Eve interception
if npr.rand() > ( 1 - eveInterceptRate ):
  eveStat['intercepted'] += 1
  if aIndex == eIndex:
    eveStat['choseRight'] += 1
  (p1, res1, collapsedState) = singletState.u_propagate(rotationOpE).measure
     ↪ ((0,), do = 'C')
else :
  collapsedState = singletState


# Instead of measuring at an angle thetaA, we rotate the state as
# the qit library does not allow for simultaneously measuring
# In an arbitrary basis and only measuring a single subsystem
(p1, res1, collapsedState) = collapsedState.u_propagate(rotationOpA).measure
   ↪ ((0,), do = 'C')
if res1 == 1:
  pIndex = 'u'
else :
  pIndex = 'd'
```

```
  # Parralel axis measured, so it contributes to the private key
  if (aIndex == 1 & bIndex == 0) | (aIndex == 2 & bIndex == 2) :
    privateKey.append(res1)

  # finalState will be the collapse from measuring qubit B after A
  (p2, res2, finalState) = collapsedState.u_propagate(rotationOpB).measure
     ↪ ((1,), do = 'C')

  if res2 == 1:
    pIndex += 'u'
  else :
    pIndex += 'd'

  if len(countMatrixIndex) == 4:
    countMatrix[countMatrixIndex][pIndex] += 1

print('All particles have been received and measured...\n')
print('Alice and Bob announce their off-axis results. Calculating the big S
   ↪ value to detect Eve.\n')
s = calcBigOlSVal(countMatrix)
expected = -2*np.sqrt(2)
ratio = np.abs( (s - expected ) / expected )
if  ratio > eveDetTol :
  print('!!!!! Eve Detected !!!!!')
  print('An S value of {0} was found. This deviation from the true value of
     ↪ {3} \nis {1} times higher than the Eve tolerance of {2}%.'.format(s,
     ↪ round(ratio/eveDetTol, 2), 100*eveDetTol, expected ))
else :
  print('An S value of {0} was found. This is within {1}% of the expected
     ↪ value {2}.\nThis is below the Eve Detection tolerance of {3}% and
     ↪ therefore and we assume no Eve presence'.format(s, 100*round(ratio, 2)
     ↪ , expected, 100*eveDetTol))

if eveInterceptRate > 0 :
  eRatio = float(eveStat['choseRight']) / eveStat['intercepted']
  print('\nEve intercepted {0} particles and chose the correct analyzer angle
     ↪ {1}% of the time.'.format(eveStat['intercepted'], 100*round(eRatio,2))
     ↪ )

print('\nThe calculated private key is {0} bits long'.format(len(privateKey)))
   ↪ ;
if debugOn:
  print('\nThe generated private key is {0}'.format(privateKey))
else :
  print('Enable debugging to see the full key.')
```

```
    print("\n=========================================================")
    print("     Simulating of the E-91 Protocol has concluded        ")
    print("=========================================================")

E91_simulate(2000);
```