Satellite Quantum Modelling & Analysis Software

Version 1.0: Documentation

J. S. Sidhu, T. Brougham, D. McArthur, R. G. Pousa and D. K. L. Oi

University of Strathclyde

Department of Physics

John Anderson Building, 107 Rottenrow East, Glasgow, G4 0NG

May 2021

Chapter 1

Introduction

1.1 Mission statement

We provide a numerical key rate analysis, which determines the amount of expected key generation in satellite-based quantum key distribution protocols. This key length analysis will help develop an intuition on the effects of different operational scenarios on the key rate and inform the development of source and receiver systems. This numerical toolkit will provide a guide to future satellite missions.

1.2 Scope of current version

The current release of SatQuMA, version 1.0, is designed to be the first working version that can calculate finite key lengths for a limited set of systems and circumstances. We implement an optimised, asymmetric two-decoy state BB84 protocol with weak coherent pulses. Elevation, and time, dependent system losses are defined externally and read-in from a data file; an example file is supplied with the software. We consider multiple satellite overpasses, but only for identical orbits. Protocol parameters can be either optimised or specified, collectively.

1.3 Installation

The latest version of the SatQuMA software can be found at `https://github.com/cnqo-qcomms/SatQuMA`.

1.3.1 Required packages

The current version of SatQuMA requires an installation of Python 3.* and the following standard packages:

- scipy (SciPy),
- numpy (NumPy),
- sys,
- time.

Chapter 2

Theoretical summary

2.1 Background

Here we present a high-level summary of the equations required to calculate the secret key length (SKL) which appear in the current SatQuMA release. We do not, however, provide any form of derivation for these relations here.

2.1.1 Protocol and statistics

In our protocol, Alice randomly prepares a state in the basis X or Z, where we usually assume $X = \{D, A\}$ and $Z = \{H, V\}$, with one of three intensities $\mu = \{\mu_1, \mu_2, \mu_3\}$, each with a probability of being selected $P_\mu = \{p_{\mu_1}, p_{\mu_2}, p_{\mu_3}\}$.

Once Alice has sent her signals to Bob and the reconciliation process, error correction, and post-processing has been completed we can define some measurement statistics from the sifted key. We define the number of events, for each basis, for each intensity Alice could prepare

$$n_{\mathrm{X},\mu} := \{n_{\mathrm{X},\mu_1}, n_{\mathrm{X},\mu_2}, n_{\mathrm{X},\mu_3}\}, \tag{2.1}$$

$$n_{\mathrm{Z},\mu} := \{n_{\mathrm{Z},\mu_1}, n_{\mathrm{Z},\mu_2}, n_{\mathrm{Z},\mu_3}\}, \tag{2.2}$$

and similarly we define the number of bit errors, for each basis, for each intensity

$$m_{\mathrm{X},\mu} := \{m_{\mathrm{X},\mu_1}, m_{\mathrm{X},\mu_2}, m_{\mathrm{X},\mu_3}\}, \tag{2.3}$$

$$m_{\mathrm{Z},\mu} := \{m_{\mathrm{Z},\mu_1}, m_{\mathrm{Z},\mu_2}, m_{\mathrm{Z},\mu_3}\}. \tag{2.4}$$

2.1.2 Secure key length

The length of the secure key is given by [1]

$$\ell = \left\lfloor s_{\mathrm{X},0} + s_{\mathrm{X},1}\left[1 - h(\phi_{\mathrm{X}})\right] - \lambda_{\mathrm{EC}} - 6\log_2\left(\frac{21}{\epsilon_{\mathrm{s}}}\right) - \log_2\left(\frac{2}{\epsilon_{\mathrm{c}}}\right)\right\rfloor, \quad (2.5)$$

where the outer brackets indicate that we should take the floor of this expression. Here, $s_{\mathrm{X},0}$ is the number of vacuum events, $s_{\mathrm{X},1}$ is the number of single-photon events, and ϕ_{X} is the phase error rate in the sifted X basis. The parameter λ_{EC} provides an estimate, or a bound, on the number of bits required for error correction although this should be replaced with the *actual* number of bits used when this is known. The security parameters ϵ_{c} and ϵ_{s} are the prescribed security parameters which define the *correctness* and *secrecy* of the resulting key respectively. The binary entropy function used above is defined as

$$h(x) = -x\log_2 x - (1-x)\log_2(1-x). \quad (2.6)$$

2.1.3 Number of vacuum events

The number of vacuum events in a particular basis is evaluated as, for example,

$$s_{\mathrm{X},0} \geq \tau_0 \frac{\mu_2 n^-_{\mathrm{X},\mu_3} - \mu_3 n^+_{\mathrm{X},\mu_2}}{\mu_2 - \mu_3}, \quad (2.7)$$

where the probability that Alice sends an n-photon state is given by the Poisson distribution,

$$\tau_n = \sum_{j=1}^{3} \frac{\mathrm{e}^{-\mu_j}\mu_j^n p_j}{n!}. \quad (2.8)$$

In order to account for statistical fluctuations in the expected number of n-photon events, we apply the Chernoff bound and define the functions [2]

$$n^+_{\mathrm{X},\mu_j} = \frac{\mathrm{e}^{\mu_j}}{p_{\mu_j}}\left[n_{\mathrm{X},\mu_j} + \log_{\mathrm{e}}\left(\frac{21}{\epsilon_{\mathrm{s}}}\right) + \sqrt{2n_{\mathrm{X},\mu_j}\log_{\mathrm{e}}\left(\frac{21}{\epsilon_{\mathrm{s}}}\right) + \log_{\mathrm{e}}\left(\frac{21}{\epsilon_{\mathrm{s}}}\right)^2}\right], \quad (2.9a)$$

$$n^-_{\mathrm{X},\mu_j} = \frac{\mathrm{e}^{\mu_j}}{p_{\mu_j}}\left[n_{\mathrm{X},\mu_j} - \frac{1}{2}\log_{\mathrm{e}}\left(\frac{21}{\epsilon_{\mathrm{s}}}\right) - \sqrt{2n_{\mathrm{X},\mu_j}\log_{\mathrm{e}}\left(\frac{21}{\epsilon_{\mathrm{s}}}\right) + \frac{1}{4}\log_{\mathrm{e}}\left(\frac{21}{\epsilon_{\mathrm{s}}}\right)^2}\right]. \quad (2.9b)$$

Note, we could also use these expressions to determine the number of vacuum events in the Z basis by exchanging the n_{X,μ_j} terms for the corresponding n_{Z,μ_j} terms.

2.1.4 Number of single-photon events

The number of single-photon events in a particular basis is similarly evaluated as,

$$s_{\mathrm{X},1} \geq \tau_1 \frac{\mu_1 \left[n^-_{\mathrm{X},\mu_2} - n^+_{\mathrm{X},\mu_3} - \frac{\mu_2^2-\mu_3^2}{\mu_1^2} \left(n^+_{\mathrm{X},\mu_1} - \frac{s_{\mathrm{X},0}}{\tau_0} \right) \right]}{\mu_1 \left(\mu_2 - \mu_3 \right) - \mu_2^2 + \mu_3^2}, \tag{2.10}$$

where τ_1 is given by (2.8). As with the expressions for the number of vacuum events, we can use the same expression to determine the number of single-photon events in the Z basis by again exchanging the n_{X,μ_j} terms for the corresponding n_{Z,μ_j} terms.

2.1.5 The phase error rate

We evaluate the phase error rate in the X basis according to

$$\phi_{\mathrm{X}} \leq \frac{v_{\mathrm{Z},1}}{s_{\mathrm{Z},1}} + \gamma \left(\epsilon_{\mathrm{s}}, \frac{v_{\mathrm{Z},1}}{s_{\mathrm{Z},1}}, s_{\mathrm{Z},1}, s_{\mathrm{X},1} \right), \tag{2.11}$$

where we use the function

$$\gamma \left(a, b, c, d \right) = \sqrt{\frac{\left(c + d \right) \left(1 - b \right) b}{cd \log_{\mathrm{e}} 2} \log_2 \left[\frac{c+d}{bcd \left(1 - b \right)} \frac{21^2}{a^2} \right]}, \tag{2.12}$$

and the single-photon events are defined as above in Sec. 2.1.4. We have also introduced the number of bit errors associated with single-photon events in Z

$$v_{\mathrm{Z},1} \leq \tau_1 \frac{m^+_{\mathrm{Z},\mu_2} - m^-_{\mathrm{Z},\mu_3}}{\mu_2 - \mu_3}, \tag{2.13}$$

where the bounds on the number of bit errors due to statistical fluctuations, $m^{\pm}_{\mathrm{Z},\mu_j}$ are given by (2.9a) and (2.9b) where the number(s) of events in the X basis, n_{X,μ_j}, should be substituted with the number(s) of bit errors in the Z basis, m_{Z,μ_j}.

2.1.6 Estimating the amount of error correction

We estimate the number of bits that need to be sacrificed to perform the error correction in two ways: the first is more accurate but also much more complex; the second is simple to implement but provides only a lower bound.

Method 1

We can evaluate the number of bits that we need to sacrifice for error correction as [3]

$$\lambda_{\mathrm{EC}} \approx n_X h\left(\mathrm{QBER}_{\mathrm{X}}\right) + \left[n_X\left(1-\mathrm{QBER}_{\mathrm{X}}\right) - F^{-1}\left(\epsilon_{\mathrm{c}}; \lfloor n_{\mathrm{X}} \rfloor, 1-\mathrm{QBER}_{\mathrm{X}}\right) - 1\right] \times \log_{\mathrm{e}}\left[\frac{\left(1-\mathrm{QBER}_{\mathrm{X}}\right)}{\mathrm{QBER}_{\mathrm{X}}}\right] - \frac{1}{2}\log_{\mathrm{e}} n_{\mathrm{X}} - \log_{\mathrm{e}}\left(\frac{1}{\epsilon_{\mathrm{c}}}\right), \tag{2.14}$$

where we define the quantum bit error rate in the X basis as

$$\mathrm{QBER}_{\mathrm{X}} = \frac{\sum_j m_{\mathrm{X},\mu_j}}{\sum_j n_{\mathrm{X},\mu_j}}, \text{ for } j \in \{1,2,3\}, \tag{2.15}$$

and $F^{-1}(k;n,p)$ is the inverse (or quantile function) of the binomial cumulative distribution function

$$F(k;n,p) = \sum_{i=0}^{\lfloor k \rfloor} \binom{n}{i} p^i \left(1-p\right)^{n-i}.$$

Method 2

Another method, based upon the block size, estimates the lower bound on the error correction as

$$\lambda_{\mathrm{EC}} \geq 1.16 \sum_{j=1}^{3} n_{\mathrm{X},\mu_j} h\left(\mathrm{QBER}_{\mathrm{X}}\right). \tag{2.16}$$

Method 3

We can simply estimate the lower bound on the error correction based upon the total number of bit errors in the X basis

$$\lambda_{EC} \geq 1.16 \sum_{j=1}^{3} m_{X,\mu_j}. \tag{2.17}$$

Chapter 3

Example of use

Here we go through the process of setting up an optimisation calculation using SatQuMA, with script excerpts taken from `SatQuMA_1.0.py` as indicated by the specified line numbers.

3.1 Optimisation parameters

SatQuMA allows the main protocol parameters to be either optimised or specified, collectively. To enable the optimisation of the protocol parameters we must set the relevant boolean flag to `True`.

```
67  tOptimise = F_or_T[1]  # False (0) or True (1)
```

The optimiser requires these parameters to be given a range, with an upper and lower bound, which we define using a `numpy` array.

```
80      xb = np.array([[0.3,1.0],[0.6,0.9999],[0.0,0.4],[0.3,1.0],[0.1,
    ↪ 0.5]])
```

Each pair of numbers (columns) of the `numpy` array are the (non-inclusive) lower and upper bounds for the parameters in order $P_x, p_1, p_2, \mu_1, \mu_2$.

We must now set initial values for the optimised parameters, keeping within the pre-defined, respective parameter ranges. These initial values can either be directly specified or they can be selected for us randomly. We note that initialising the parameters can be an important step as the returned secret key length (SKL) can be zero over relatively broad parameter regions, or indeed in smaller localised regions, which may cause the optimiser to report spurious issues with the supplied function (that it's derivatives appear to be zero). In order to specify the initial values, we first set the relevant boolean flag to be `True`.

```
90      tInit = F_or_T[1]  # False (0) or True (1)
```

Next, we set the values for each parameter.

```
96          Px_i  = 0.5  # Asymmetric polarisation probability
97          pk1_i = 0.7  # Probability Alice prepares intensity 1
98          pk2_i = 0.1  # Probability Alice prepares intensity 2
99          mu1_i = 0.8  # Intensity 1
100         mu2_i = 0.3  # Intensity 2
```

3.2 Calculation parameters

3.2.1 Input file options

SatQuMA requires a time/elevation *vs* link efficiency data file (in CSV format) to be specified, hereafter referred to as the 'loss file'. An example file has been supplied with the software, the name of which we must specify.

```
127 loss_file = 'FS_loss_XI0.csv'
```

We can also specify the directory containing the loss file, if it is not located in the current working directory. If the loss file is in the work directory, then we can set this path to be an empty string.

```
129 loss_path = ''
```

Additionally, we can specify the column within the loss file that contains the link efficiency (losses), however the default is column 3.

```
130 lc        = 3 # Column containing loss data in file (counting from 1
    ↪ )
```

3.2.2 Output file options

During a calculation, SatQuMA can write to two different output streams: a local file and the standard output (IDE/terminal/screen/*etc*). To request that calcualtion data (and metrics) are written to file (in CSV format) we first set the relevant boolean flag to be `True`, then choose the path and filename for that output.

```
136 tWriteData = F_or_T[1]  # False (0) or True (1)
137 outpath    = ''    # Path for output file (empty = current directory
    ↪ )
138 outfile    = 'out' # Name for output file (minus .csv)
```

We can further request that SatQuMA provide a sorted output file: data rows are sorted by SKL per relative system loss (see 3.2.3).

```
140 tSortData  = F_or_T[1]  # False (0) or True (1)
```

This sorted output file can also be 'optimised' according to time window (dt), which returns only the maximum SKL for each system loss value (*i.e.* for only the optimum calculated time window).

```
142 tOptiData  = F_or_T[1] # False (0) or True (1)
```

Finally, we request that SatQuMA print out data to the standard output stream as they are calculated.

```
144 tPrint     = F_or_T[1]  # False (0) or True (1)
```

3.2.3 Time window and system loss

The first version of SatQuMA has been designed to calculate (loop) over the duration of the overpass time half-window (s) and the relative system loss (dB). The time window loop is principally controlled by a `numpy` array specifying the start, stop and step indices relating to the time slots (as specified in the input loss file, see 3.2.1).

```
151 dt_range = np.array([201, 221, 10]) # Start, stop, step
```

Note, these time slots are labelled relative to the overpass zenith for which we arbitrarily set $t = 0$. Here we have requested SKL calculations with overpass transmit time half-windows of, initially, up to 201 s then rising to half-windows of a duration of 221 s in 10 s increments. We can also set a minimum elevation for transmission (in degrees) which will override the values we have just specified where necessary.

```
154 min_elev = 10.0 # Minimum elevation transmission angle (degs)
```

Here, we have specified that no transmission is possible for elevations below 10°.

Next, we define the start, stop and step values for a loop over the excess system losses. That is, we can add additional loss to those specified in the loss file (converting from system efficiency).

```
157 ls_range = np.array([0, 12, 2])   # Start, stop, step value
```

Here, we have considered systems which have 12 to 13 dB of excess loss above the losses specified in the input loss file.

3.2.4 System parameters

We now specify the relevant parameters which characterise the performance of the system we wish to model. First, we specify the orbit offset angle ξ (rad.) which define the smallest angle between the orbit plane of the satellite and the zenith plane of the optical ground station (OGS).

```
163 xi  = 0.0 # Angle between OGS zenith and satellite (from Earth's
     ↪ centre) [rad.]
```

This value should be the same as that used when producing the input loss file, as such we typically include the value of ξ in the name of this file. At present, however, this value is simply included in the output data and doesn't factor into the calculations directly.

Next, we specify the remaining protocol parameters: the intensity of the third weak coherent pulse (second decoy state) μ_3 and the prescribed errors in both correctness and secrecy, ϵ_c and ϵ_s respectively.

```
165 mu3   = 0          # Intensity of pulse 3 (fixed)
167 eps_c = 10**(-15) # Correctness parameter
168 eps_s = 10**(-9)  # Secrecy parameter
```

Here, by setting $\mu_3 = 0$ we have chosen the third pulse in our protocol to be the vacuum state.

The after-pulse probability of each detector, percentage error due to polarisation misalignment (or intrinsic quantum bit error rate, $QBER_{\mathrm{I}}$), and dark count probability are now specified.

```
170 Pap = 0.001        # After-pulse probability
172 PolError = 0.005  # Polarization error (QBER_I)
174 Pdc = 5*10**(-7)  # Dark count probability
```

Finally, we specify the number of (identical) satellite overpasses to include and the repetition rate of the transmission source (Hz).

```
176 NoPass = 1         # Number of satellite passes
178 Rrate  = 1*10**(8) # Source rate (Hz)
```

3.2.5 Advanced parameters

We can also specify some of the advanced parameters; these parameters have been set to default values which should only be changed by more advanced users looking for additional control over the calculations as they may strongly affect the software performance and resulting SKL. We will specify that the protocol uses the Chernoff bounds, for all tail bounds, and that the SKL be optimised both including and excluding the estimate of error correction (in the latter case, error correction is estimated post-optimisation).

```
198 tChernoff = F_or_T[1]  # False (0) or True (1)
210 tCompareEC  = F_or_T[1]  # False (0) or True (1)
```

We note that when comparing the effect of including the estimate of the number of bits required for error correction both during and after optimisation only the former case is included in the output data.

3.3 Visualising data

Once the software has successfully completed the calculation we can visualise the output. Here we will focus on the optimised, sorted output data file `out_opt.csv`. For example, we can plot the secret key length as a function of the total system loss, both including and excluding the error correction estimation, using the following (minimum working) python code.

```
import numpy as np
data = np.loadtxt('out_opt.csv',skiprows=1,delimiter=',')
x   = data[:,0]      # Total system loss
y1  = data[:,2]      # SKL
y1_ = y1 + data[:,7] # SKL + lambda_EC
import matplotlib.pyplot as plt
fig1, ax1 = plt.subplots(1,1,figsize=(5,5))
ax1.semilogy(x,y1,'-',x,y1_,'--')
```

The resulting graph is shown in Fig. 3.1(a). We may also extend our plotting code to visualise the error rates associated with these finite keys, as shown in Fig. 3.1(b).

```
fig2, ax2 = plt.subplots(1,1,figsize=(5,5))
y2  = data[:,3:5]    # QBERx and phi_x
ax2.plot(x,y2[:,0],'-',x,y2[:,1],'--')
```

Finally, we may also plot the change in the optimised protocol parameters, as shown in Fig. 3.2, using the following code.

```
fig3, ax3 = plt.subplots(1,1,figsize=(5,5))
y3  = data[:,20:27]  # Protocol parameters
ls  = [':','-','-','-','--','--','--'] # Linestyles
for ii in range(0,7,1):
    ax3.plot(x,y3[:,ii],ls[ii])
```

The various other parameters and output data from this file are listed in Table 3.1.

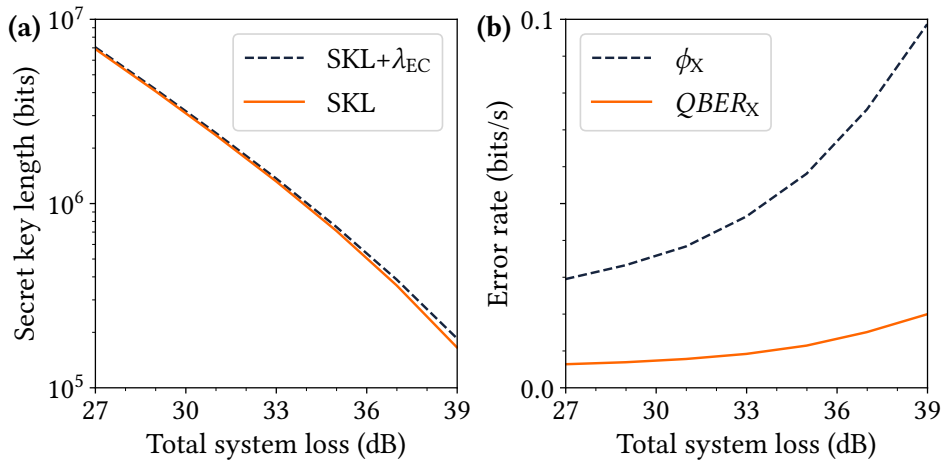

Figure 3.1: Total system loss in decibels against (a) secret key length, with and without error correction estimation, and (b) the phase and quantum bit error rates for the X basis.

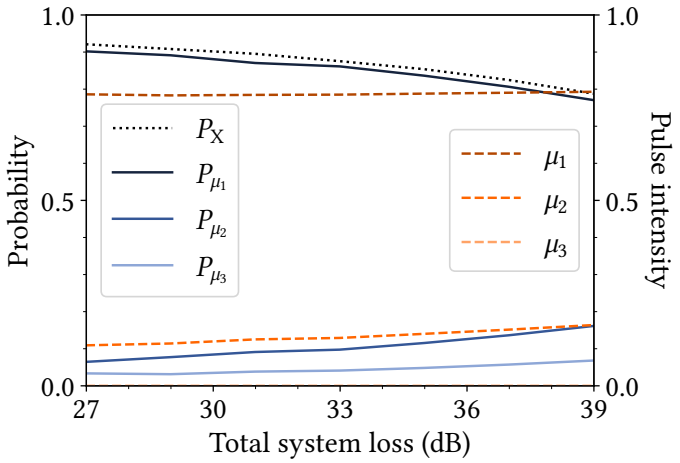

Figure 3.2: Optimised protocol parameters (for asymmetric BB84) as a function of the total system loss in decibels. Note, $\mu_3 = 0$.

Index	Variable	Symbol	Description
0	ls+sysLoss	-	Total system loss (dB).
1	dt	dt	Transmission half-window duration (s).
2	SKL	ℓ	Secret key length (bits).
3	QBERx	$\mathrm{QBER_X}$	Quantum bit error rate for X basis (bits/s).
4	phi_x	ϕ_{X}	Phase error rate for X basis (bits/s).
5	nX	n_{X}	Number of events in the X basis.
6	nZ	n_{Z}	Number of events in the Z basis.
7	lambdaEC	λ_{EC}	Estimate of number of bits used for error correction.
8	sX0	$s_{\mathrm{X},0}$	No. of vacuum events for X basis.
9	sX1	$s_{\mathrm{X},1}$	No. of single photon events for X basis.
10	vz1	$v_{\mathrm{Z},1}$	No. of bit errors associated with single-photon events in Z basis.
11	sZ1	$s_{\mathrm{Z},1}$	No. of single photon events for Z basis.
12	mpn	$\sum_j \mu_j/3$	Mean transmitted photon number.
13	PolError	$\mathrm{QBER_I}$	Intrinsic quantum bit error rate (%).
14	Pdc	P_{dc}	Probability of dark count event.
15	Pap	P_{ap}	Probability of after-pulse event.
16	NoPass	M	Number of satellite overpasses.
17	Rrate	N	Source repetition rate (Hz).
18	eps_c	ϵ_c	Correctness parameter.
19	eps_s	ϵ_s	Secrecy parameter.
20	Px	P_{X}	Polarisation bias for X basis.
21	P1	P_{μ_1}	Probability of sending pulse 1.
22	P2	P_{μ_2}	Probability of sending pulse 2.
23	P3	P_{μ_3}	Probability of sending pulse 3.
24	mu1	μ_1	Intensity of pulse 1.
25	mu2	μ_2	Intensity of pulse 2.
26	mu3	μ_3	Intensity of pulse 3.
27	xi	ξ	Offset angle of satellite orbital plane from OGS zenith (deg).
28	min_elev	$\theta_{\min}$	Minimum elevation of satellite for transmission (deg).
29	max_elev	$\theta_{\max}$	Maximum elevation of satellite overpass (deg).

Table 3.1: Table of data written to main output file(s) giving the index (data column), python variable name, associated mathematical symbol and a brief description.

Chapter 4

Bugs and future releases

SatQuMA is still very much under development with improved versions planned for release in the near future. The current version employs an out-of-the-box optimisation algorithm, `trust-constr` from scipy.

4.1 Known bugs

Unfortunately, as the function being minimized does not have derivatives that can be continuously defined for all regions of the relevant parameter space the scipy optimiser can report errors when it encounters locally flat regions of the function space, typically where the SKL drops to zero.

4.2 Reporting bugs

If you encounter any behaviour you consider unexpected, or anytime python raises an exception (excluding for user error), please send a copy of the main SatQuMA python file, and include any relevant output files, to the development team at the University of Strathclyde. It is always good practice to add python files to an archive (zip/tar/etc) before sending via email to avoid mail being rejected by mail scanning algorithms.

4.3 Suggested content

If there are any features that you would like to see added to this software then please feel free to contact the development team at the University of Strathclyde.

Bibliography

[1] C. C. W. Lim, M. Curty, N. Walenta, F. Xu, and H. Zbinden, "Concise security bounds for practical decoy-state quantum key distribution," *Phys. Rev. A*, vol. 89, p. 022307, February 2014.

[2] H.-L. Yin, M.-G. Zhou, J. Gu, Y.-M. Xie, Y.-S. Lu, and Z.-B. Chen, "Tight security bounds for decoy-state quantum key distribution," *Sci. Rep.*, vol. 10, p. 14312, August 2020.

[3] M. Tomamichel, J. Martinez-Mateo, C. Pacher, and D. Elkouss, "Fundamental finite key limits for one-way information reconciliation in quantum key distribution," *Quant. Inf. Proc.*, vol. 16, p. 280, October 2017.