# Configuration Manual

MSc Research Project
MSc Cyber Security


Sonal Hajare
Student ID: x21132372

School of Computing
National College of Ireland


Supervisor: Prof. Imran Khan

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

| | |
|---|---|
| **Student Name:** | Sonal Nanaji Hajare ………………………………………………………………………………………………… |
| **Student ID:** | X21132372 ……………………………………………………………………………………………….…… |
| **Programme:** | MSc. Cyber Security … ………………………………………………………… **Year:** 2022 - 2023 ………………………….. |
| **Module:** | Academic Internship ……………………………………………………………………………………….……… |
| **Lecturer:** | Imran Khan ……………………………………………………………………………………………….……… |
| **Submission Due Date:** | 01-02-2023 ……………………………………………………………………………………………….……… |
| **Project Title:** | Securing UAV communication using Quantum Cryptography ……………………………………………………………………………………………….……… |
| **Word Count:** | …………1405……………………………… **Page Count:** ………14…………… |

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**Signature:** Sonal Nanaji Hajare ………………………………………………………………………………………………………

**Date:** 28-1-2023 ………………………………………………………………………………………………………

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

| | |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies) | □ |
| **Attach a Moodle submission receipt of the online project submission,** to each project (including multiple copies). | □ |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | □ |

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Configuration Manual

Sonal Hajare
Student ID: x21132372

## 1 Configuration requirements

Install Qutip, NumPy, SciPy, and Jupyter.

```
C:\Windows\System32>pip install qutip
WARNING: Ignoring invalid distribution -ip (c:\python310\lib\site-packages)
WARNING: Ignoring invalid distribution -ip (c:\python310\lib\site-packages)
Collecting qutip
  Using cached qutip-4.7.0-cp310-cp310-win_amd64.whl (5.4 MB)
Collecting scipy>=1.0
  Using cached scipy-1.9.1-cp310-cp310-win_amd64.whl (38.6 MB)
Collecting packaging
  Using cached packaging-21.3-py3-none-any.whl (40 kB)
Requirement already satisfied: numpy>=1.16.6 in c:\python310\lib\site-packages (from qutip) (1.23.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\python310\lib\site-packages (from packaging->qutip) (3.0.9)
WARNING: Ignoring invalid distribution -ip (c:\python310\lib\site-packages)
Installing collected packages: scipy, packaging, qutip
WARNING: Ignoring invalid distribution -ip (c:\python310\lib\site-packages)
WARNING: Ignoring invalid distribution -ip (c:\python310\lib\site-packages)
WARNING: Ignoring invalid distribution -ip (c:\python310\lib\site-packages)
Successfully installed packaging-21.3 qutip-4.7.0 scipy-1.9.1
```

**Figure 1: Shows the installation**

Creating a conda environment for Qutip

```
Anaconda Prompt (anaconda3)

(base) C:\Users\MyPc>conda create -n qutip-env python
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

**Figure 2: Shows command creating conda environment**

Verifying the installation

```
Python 3.10 (64-bit)
Python 3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import qutip.testing
C:\Python310\lib\site-packages\qutip\__init__.py:96: UserWarning: matplotlib not found: Graphics will not work.
  warnings.warn("matplotlib not found: Graphics will not work.")
>>> qutip.testing.run()
```

**Figure 3: Shows the successful installation**

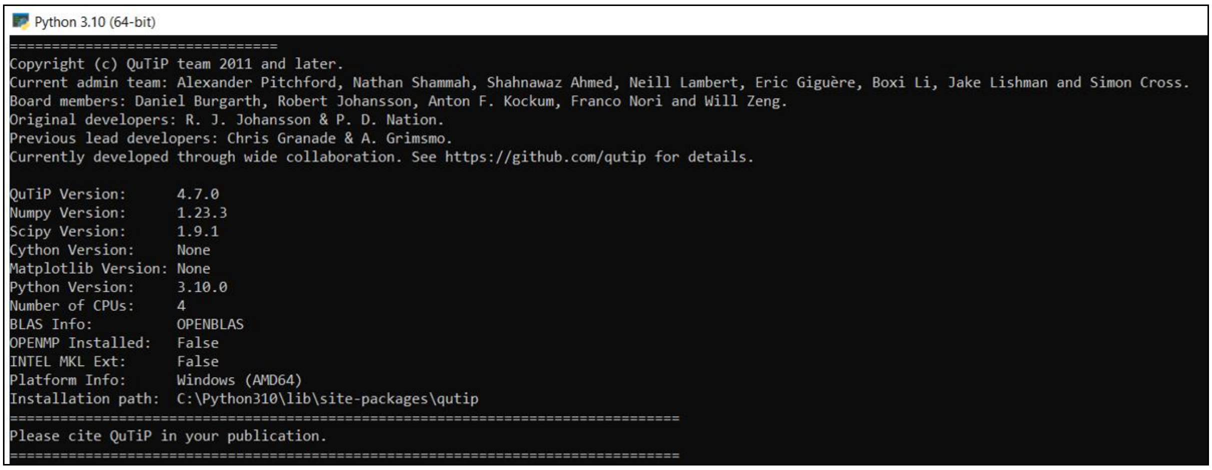


**Figure 4: Shows the successful installation**

Starting jupyter notebook

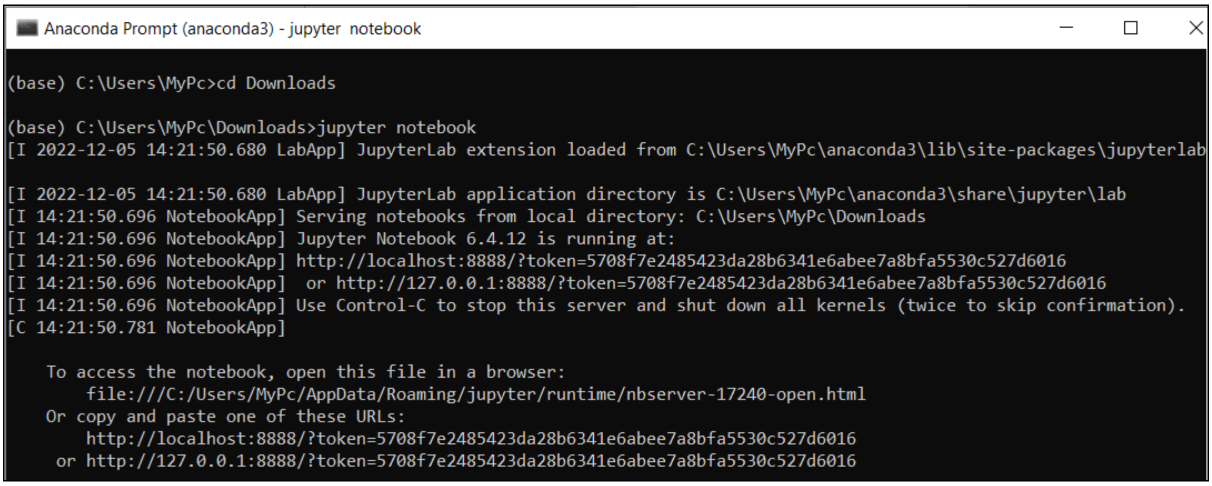


**Figure 5: Shows the command to start the jupyter notebook**

An automated new tab will open in the browser. According to the needs of the project, scripts are modified. All the scripts and outcomes are shown here as an example. The detection of EVE and the time required for it are determined by one script, and the generation of the secret key and the time needed for that are shown by the same script with different settings. (Set presence of EVE = “False”). In addition, the script uses the RC6 technique to encrypt and decrypt plain text.

Following the entry of different bits—128bit, 192bit, and 256bit—with and without EVE, the same script was saved many times to display varied results for better understanding.

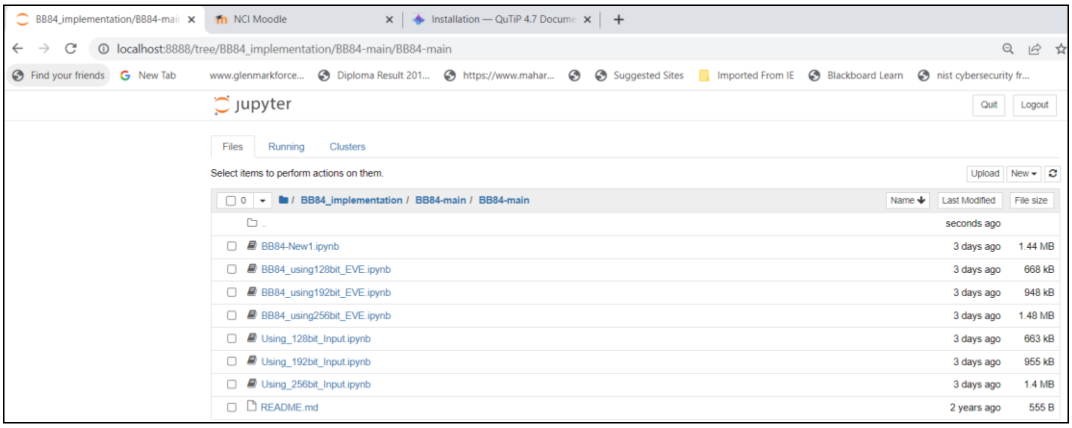


**Figure 6: Shows the jupyter tab in the browser**

# 2 Detecting EVE

Open the script and make sure "Eve presence" should be set to "True".

```
# Determine whether eavesdropping will take place
eavesdropper_present = True
```

**Figure 7: Setting EVE presence as True**

Run the first block of the script and it will ask to provide the initial input.

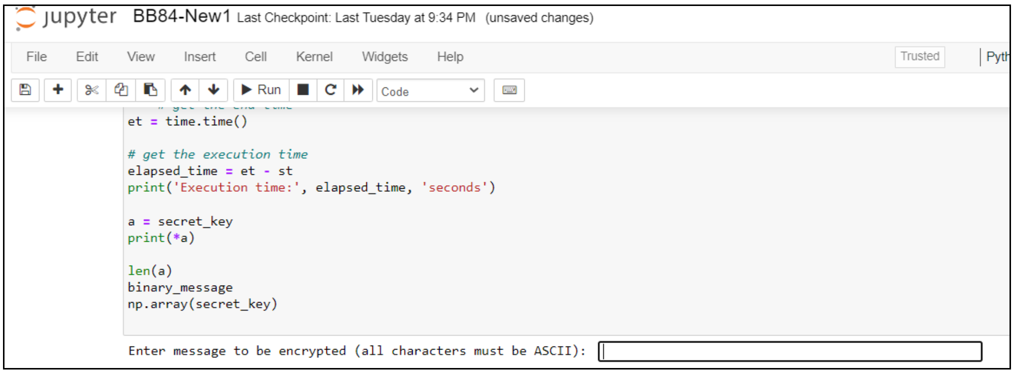


**Figure 8: Shows the script asking for the initial input message**

Provide the 128-bit input and wait till it shows the presence of EVE and calculates the execution time.
Initially, the script will check whether Alice and Bob can create a secret key by executing all the phases.

```
Enter message to be encrypted (all characters must be ASCII): bQeThVmYq3t6w9z$
```

**Figure 9: Shows the given 128-bit input message**

Alice and Bob define the constants for rectilinear and diagonal basis during the preparation stage. The script will then produce m-size random base sequences for both.

```
# Determine message length and the lenth of the random sequances
n = len(binary_message)
m = 6*n
```

**Figure 10: Shows the calculation of m with respect to n**

To encourage Bob's selection of polarization filters, vertical and diagonal filters are also defined in this phase as measurement operators.

```
# 1) Preparation phase

#  Define the constants that Bob and Alice agree on in the preparation phase
RECTILINEAR_BASIS = 0
DIAGONAL_BASIS = 1

# In rectilinear basis
HORIZONTAL_POL = 0
VERTICAL_POL = 1

# In diagonal basis
DIAGONAL_45_POL = 0
DIAGONAL_135_POL = 1

# Generate Alice's and Bob's random bases sequences of size m
alice_rand_bases_seq = np.random.choice([RECTILINEAR_BASIS, DIAGONAL_BASIS], size=m)
bob_rand_bases_seq = np.random.choice([RECTILINEAR_BASIS, DIAGONAL_BASIS], size=m)

# Generate Alice's random bit sequence of size m
alice_rand_bit_seq = np.random.choice([0, 1], size=m).tolist()

if eavesdropper_present:
    # Generate Eve's random bases sequence of size m
    eve_rand_bases_seq = np.random.choice([RECTILINEAR_BASIS, DIAGONAL_BASIS], size=m)

alice_rand_bases_seq
bob_rand_bases_seq
np.array(alice_rand_bit_seq)
```

**Figure 11: Shows the Preparation phase**

OUTPUT: Below figures show the random base sequences generated by Alice and Bob.

```
alice_rand_bases_seq

array([1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
       0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0,
       0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
       1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0,
       1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1,
       0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0,
       0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1,
       0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1])
```

**Figure 12: Shows the random bases sequences generated by Alice**

```
bob_rand_bases_seq
```

```
array([1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1,
       1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1,
       0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1,
       1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1,
       0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0,
       0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1,
       0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0,
       1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0,
       1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1,
       0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0,
       0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1])
```

**Figure 13: Shows the random bases sequences generated by Bob**

```
# Describe bases of Hilbert vector space
basis_0 = qt.basis(2,0)
basis_1 = qt.basis(2,1)

# Describe polarization states in Hilbert vector space
photon_h = basis_0                              # horizontally polarized photon
photon_v = basis_1                              # vertically polarized photon
photon_d45 = (basis_0 + basis_1).unit()         # diagonally polarized photon (45 deg)
photon_d135 = ((-1)*basis_0 + basis_1).unit() # diagonally polarized photon (135 deg)

photon_h
photon_v

photon_d45
photon_d135
```

**Figure 14: Shows the Hilbert vector space polarization states**

OUTPUT: Below figure shows the quantum objects as per Hilbert vector space bases.
In figure 16, we can see the vertically polarized photon, horizontally polarized photon, and diagonally polarized photons in green, orange, and blue color respectively.

```
photon_h
```

Quantum object: dims = [[2], [1]], shape = (2, 1), type = ket
$$\begin{pmatrix} 1.0 \\ 0.0 \end{pmatrix}$$

```
photon_v
```

Quantum object: dims = [[2], [1]], shape = (2, 1), type = ket
$$\begin{pmatrix} 0.0 \\ 1.0 \end{pmatrix}$$

```
photon_d45
```

Quantum object: dims = [[2], [1]], shape = (2, 1), type = ket
$$\begin{pmatrix} 0.707 \\ 0.707 \end{pmatrix}$$

```
photon_d135
```

Quantum object: dims = [[2], [1]], shape = (2, 1), type = ket
$$\begin{pmatrix} -0.707 \\ 0.707 \end{pmatrix}$$

**Figure 15: Shows the output of Hilbert vector space polarization states**

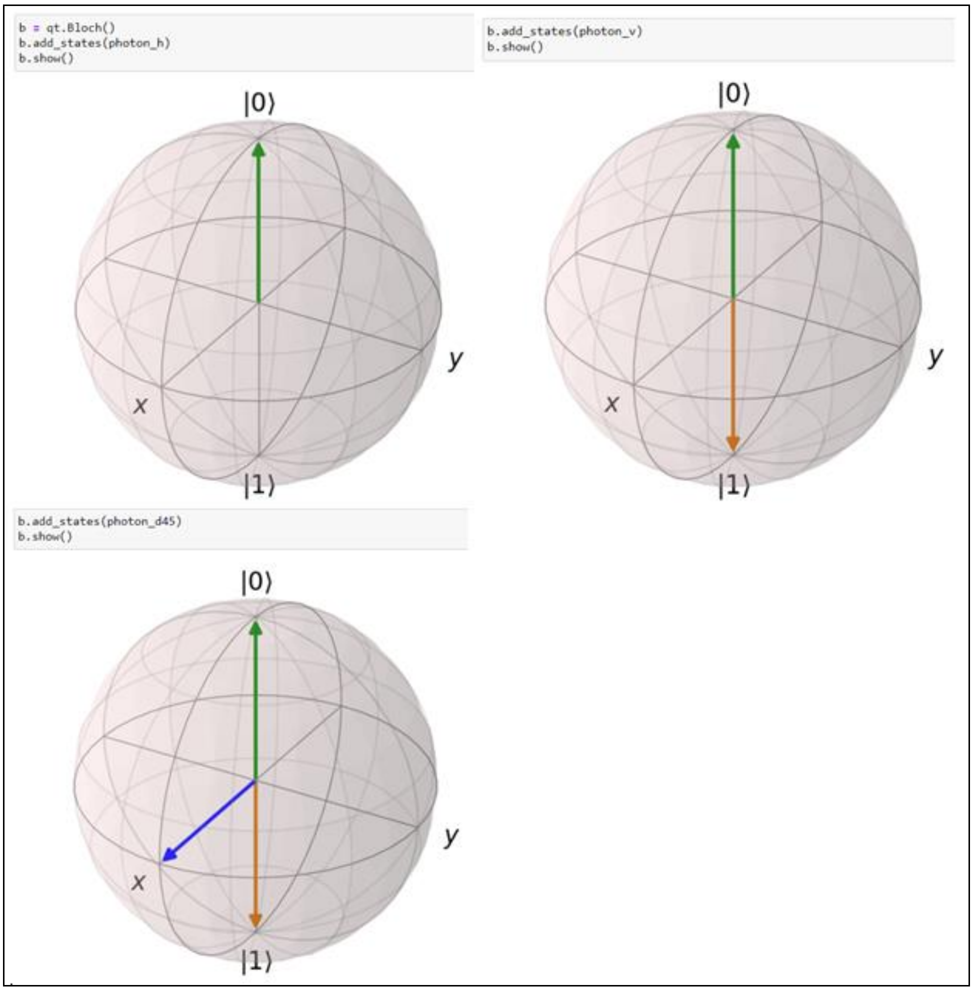


**Figure 16: Shows the graphical representation of Hilbert vector space polarization states**

The second step is the transmission phase, during which Bob keeps track of the photons Alice sent while she sent them at random polarizations as a demonstration.

```
# Perform transmission

bob_measured_values = []
photons_sent = [] # keep track of the photons Alice sent (for demonstration purposes)

for basis_a, bit_value, basis_b, i in zip(alice_rand_bases_seq, alice_rand_bit_seq, bob_rand_bases_seq, range(m)):

    # Alice picks a polarized foton source according to her random sequances
    photon, sign = pick_photon_polarization(basis_a, bit_value)
    photons_sent.append(sign)

    # Alice sends the picked photon to Bob
    if eavesdropper_present:
        _, photon = measure_polarization(photon, eve_rand_bases_seq[i])

    #Bob measures the photon
    value, _ = measure_polarization(photon, basis_b)
    bob_measured_values.append(int(value))  # append value to the end of Bob's measurements sequence

np.vstack([
          np.array(photons_sent),
          bob_rand_bases_seq,
          np.array(bob_measured_values)
          ]).T[:11, :]
```

**Figure 17: Shows the transmission phase**

OUTPUT: Below figure shows the transmission phase where Alice and Bob measured the photons sent by each other.

```
array([['D45', '1', '0'],
       ['H', '0', '0'],
       ['D135', '0', '0'],
       ['V', '0', '0'],
       ['H', '0', '0'],
       ['D45', '0', '0'],
       ['D135', '0', '1'],
       ['D45', '1', '0'],
       ['V', '0', '1'],
       ['D135', '0', '1'],
       ['H', '1', '0']], dtype='<U11')
```

**Figure 18: Shows the output of the transmission phase**

The next step is the elimination phase, where Alice and Bob compare their random base sequences and eliminate any items that were measured in the wrong base.

```
# 3) Elimination phase

# Alice and Bob compare their random bases sequances
bases_disagreement_indices = np.where(alice_rand_bases_seq != bob_rand_bases_seq)[0]

bases_disagreement_indices[:100] # See sample of the indices Bob and ALice will have to remove

# Bob removes elements which he measured in the incorrect base from his measurements sequence
for i in np.flip(bases_disagreement_indices):
    bob_measured_values.pop(i)

# ALice removes those elements from her random bit sequence
for i in np.flip(bases_disagreement_indices):
    alice_rand_bit_seq.pop(i)

len(alice_rand_bases_seq)
len(bob_rand_bases_seq)
len(alice_rand_bit_seq)
len(bob_measured_values)
```

**Figure 19: Shows the elimination phase**

OUTPUT: Below figure shows the unmatched indices removed by Alice and Bob from random bit sequences in the elimination phase.

```
bases_disagreement_indices[:100] # See sample of the indices Bob and ALice will have to remove

array([  2,   5,   6,   9,  10,  14,  16,  18,  22,  25,  28,  29,  30,
        32,  34,  35,  38,  39,  43,  44,  46,  47,  50,  51,  52,  54,
        58,  61,  62,  64,  65,  66,  68,  70,  72,  74,  75,  76,  82,
        83,  84,  85,  87,  88,  95,  97,  98,  99, 100, 101, 106, 107,
       109, 110, 112, 113, 114, 116, 117, 118, 119, 121, 122, 123, 127,
       128, 129, 130, 131, 132, 135, 138, 139, 141, 144, 145, 146, 147,
       149, 151, 155, 156, 157, 158, 162, 164, 165, 166, 169, 170, 171,
       173, 174, 175, 176, 177, 178, 179, 180, 181], dtype=int64)
```

**Figure 20: Shows removed indices in the elimination phase**

In the following phase, known as the error-check phase, Bob makes the indices available so that Alice can choose the same elements from her sequence. This step required sacrificing a greater portion of the original sequence's bits, which is why "m" was set to be six times larger than "n" in the beginning.

```
# 4) Error check phase

# Bob picks random subset of his measured values sequence, 1/3 of the sequence lenth (after elimination phase) long
error_check_indices = np.random.randint(0, len(bob_measured_values), len(bob_measured_values)//3)

error_check_indices  # Bob makes indices public for Alice to pick the same elements from her sequence

bob_error_check_subset = []
alice_error_check_subset = []

for i in np.flip(np.sort(error_check_indices)):
    bob_el = bob_measured_values.pop(i)
    bob_error_check_subset.append(bob_el)

    alice_el = alice_rand_bit_seq.pop(i)
    alice_error_check_subset.append(alice_el)

    m


    len(bob_measured_values)  # see that a big part of bits from the original sequence had to be sacrificed,
                              # that's why m was chosen 6 times bigger that n in the begining

    print(bob_measured_values)

    len(alice_rand_bit_seq)

    print(alice_rand_bit_seq)
```

**Figure 21: Shows the error check phase**

OUTPUT: Below figure shows all the indices which Bob measured and make public so that Alice picks the same elements from her sequence.

```
error_check_indices  # Bob makes indices public for Alice to pick the same elements from her sequence

array([ 75, 236, 248,  90,  20, 303, 202, 136,  37,  74, 150, 229, 213,
        39, 363, 108, 101, 316, 235, 120, 133, 353,  24, 245, 148, 316,
        86, 144,  32, 262,  73,  44, 256, 142, 116, 342,  34, 202, 191,
        66, 326, 272, 140,  58, 327, 201, 364, 254, 333,  50, 213, 342,
       224, 140, 362, 177, 111,  11, 223, 168, 347, 111, 248, 258,  63,
       147,  23, 188,  88,  70,   1,  62, 278,  91, 174, 274,  27, 293,
       345, 358, 142,   4, 358, 281, 251, 147, 110, 339, 108,  39, 224,
        88, 293,  46, 269, 210,  23, 316, 340,  68,  70, 320, 100, 332,
       314, 144, 137, 289, 157, 121, 139, 200, 364, 260, 134,  82,  94,
       222, 363, 119, 247, 333, 299])
```

**Figure 22: Shows the indices which Bob must make public for Alice**

The presence of EVE can therefore be inferred if the sequences are different from one another.

The script will determine how long it will take to run to find EVE.

```python
    # Compare chosen subsets
sequences_identical = bob_measured_values == alice_rand_bit_seq

sequences_identical

if sequences_identical:
    secret_key = alice_rand_bit_seq[:n]  # use first n bits of final sequance as key
    print("Key was safely established.")

else:
    print("Key was not safely established.")
    et = time.time()
    elapsed_time = et - st
    print('Execution time:', elapsed_time, 'seconds')
    raise SystemExit("Eavesdropper was detected! Key couldn't be safely established.")
    # The bellow code is not executed, communication has to be repeated until a safe key is established.

    # get the end time
et = time.time()

# get the execution time
elapsed_time = et - st
print('Execution time:', elapsed_time, 'seconds')
```

**Figure 23: Shows the code to compare chosen subsets**

OUTPUT: Below figure shows the measured indices by Alice and Bob and figures out whether those are identical or not. In this case, the sequences are not identical, hence the presence of EVE is captured and the execution time to capture the presence of EVE is printed.

```
print(bob_measured_values)

[0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1,
1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1,
1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1,
0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0,
1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0,
0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1,
0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0,
0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0]
```

**Figure 24: Shows the Bob measured values**

```
print(alice_rand_bit_seq)

[0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1,
1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1,
1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1,
0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0,
0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1,
1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1]
```

**Figure 25: Shows Alice measured values**

```
sequences_identical

False
```

**Figure 26: Shows the output of measured sequences**

```
Key was not safely established.
Execution time: 4.498515605926514 seconds

An exception has occurred, use %tb to see the full traceback.

SystemExit: Eavesdropper was detected! Key couldn't be safely established.
```

**Figure 27: Shows the presence of EVE and execution time**

# 3 Generating Secret key

In the script, we need to set the “eavesdropper present = false” to generate the secret key.

```
# Determine whether eavesdropping will take place
eavesdropper_present = False
```

**Figure 28: Setting EVE presence to false**

To generate the secret key and display the execution time needed to generate it, provide the 128-bit input, wait for it to complete each phase of the BB84 protocol, and make sure that EVE is not present.

```
Enter message to be encrypted (all characters must be ASCII): bQeThVmYq3t6w9z$
```

**Figure 29: Shows the provided initial message input**

Let's say that Alice and Bob need to compare their chosen subsets after going through the preparation phase, transmission phase, elimination phase, and error check phase as discussed in the above section.

OUTPUT: Below figure shows the measured values by Alice and bob.

```
print(bob_measured_values)

[1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0,
1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1,
0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0,
0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0,
1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1,
0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1,
1, 1, 0]
```

**Figure 30: Shows Bob measured values**

```
print(alice_rand_bit_seq)

[1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0,
1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1,
0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0,
0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0,
1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1,
0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1,
1, 1, 0]
```

**Figure 31: Shows the Alice measured values**

If the sequences are identical, then the key will be safely established.

OUTPUT: In this case the measured values by Alice and Bob are identical. Hence key was safely established, and time taken to generate the secret key is printed.

```
Key was safely established.
Execution time: 2.3447961807250977 seconds
```

**Figure 32: Shows the safely established key and execution time**

# 4 Providing Plain text and secret key to RC6 for encryption and decryption

Once the key has been created, the RC6 technique will be used to further process it so that it may be used to encrypt and decrypt plain text.
The RC6 algorithm requires that we give it the plain text message to encrypt and decrypt. It will then retrieve the secret key and begin the RC6 algorithm procedure to encrypt and decrypt that plain text. The plain text can be seen in the screenshot below: “Now you can share the secret data on this secure channel”.

```python
    def encription(self, message, key, w = 32, r = 20):
        self._generate_keysTable(key, w = 32, r = 20)

        while len(message) % (4 * w) != 0:
            message += '0'
        message = [message[(block * 4 * w): ((block + 1) * 4 * w)] for block in range(int(len(message) / (4 * w)))]

        output = ''
        for block in message :
            output += self._encription_binBlock(block, w, r)
        return output

    def decription(self, message, key, w = 32, r = 20):
        self._generate_keysTable(key, w = 32, r = 20)

        message_arr = [message[x * w * 4 : (x + 1) * w * 4] for x in range(int(len(message) / (w * 4)))]
        ending = message[len(message_arr) * 4 * w:]

        output = ''
        for block in message_arr:
            output += self._decription_binBlock(block, w = 32, r = 20)

        return output + ending

def test():
    message = base64.b64encode(bytes("""Now you can share the secret data on this secure channel""", 'utf-8'))
    # message = bytes("""It is just a testing.""", 'C4LwS74g3K4t45KGWK25XA==')
    key = base64.b64encode(bytes(listToString(secret_key), 'utf-8'))
    # key = bytes("""Secret key""", 'C4LwS74g3K4t45KGWK25XA==')


    cipher = Coder()
    bin_massage = cipher.bytesToBin(message)
    bin_key = cipher.bytesToBin(key)

    encription_bin_message = cipher.encription(bin_massage, bin_key) # Шифрование
    decription_bin_message = cipher.decription(encription_bin_message, bin_key) # Расшифровка
    decription_message = cipher.binToBytes(decription_bin_message)
```

**Figure 33: Shows the plain text and secret key from bb84 provided to RC6 algorithm**

OUTPUT: Below figure shows the output of encrypted messages and decrypted messages when plain text and a secret key are provided.

```
test()

MESSAGE: b'Now you can share the secret data on this secure channel'
KEY: b'MTEwMDExMTExMDAxMDAxMDAxMTExMTAxMDExMTAwMDAwMDAxMDExMTExMDExMDAxMDEwMTAwMDEwMTExMTEwMDAxMDExMDExMTExMDExMDExMDExMTExMDAw
MDEwMDAwMTAxMDAxMTAxMDExMDAwMDExMTAwMTExMDAwMDAxMDA='

BIN MESSAGE: 0101010001101101001110010011001101001001010010000110110001110110011001000101001101000010011010100101100101010111001
1101000110011101100011001100100110100001101000011000110110110101010101011001110110010001000111011010000110110001001001010010000
1001110011011000101100100110011010010100110110001100100010000110100001001101011010110010101100001010010011010000100100101000111
0011100101110101010010010100100001010010011011110110000101011000010011010110011101100011001100100101011001101010011001000101100
0010010100110110001001001010001110100111001101111010110010101011100110101011101010101101001010111011101110011110
1

ENCRIPTION BIN MESSAGE: 0001010011111001110101110010011011101000010011101011001010011010000100100010010001010111010101110101101
0110100000001110111000010010101000000100010100101101101101110011001100101011011101100100011110111011010101001011100100110100110
1001011010000101111100010100100010110001000111001100110110011111110111101111001110000000000101000101111010100001000111101111000
0
1011100010010100001111111100110100000001111011010101101000101000110111010001110011100010101000111011001110111010101011100110011
0000010010001001100010101101100100111010111010111110101101110101011010100111001110010011100111111011100011011001011001000101110
011100011110111101010011111001

DECRIPTION BIN MESSAGE: 0101010001101101001110010011001101001001010010000110110001110110011001000101001101000010011010100101100
1010101110011010001100111011000110011001001101000011010000110001101101101010101010110011101100100010001110110100001101100010010
0101001000010011100110110001011001001100110100101001101100011001000100001101000010011010110101100101011000010100100110100001001
0010100011100111001011101010100100101001000010100100110111101100001010110000100110101100111011000110011001001010110011010100110
0100010110000100101001101100010010010100011101001110011011110101100101010111001101010111010101011010010101110111011100111101000
00000000000000000000000000000

DECRIPTION MESSAGE: b'Now you can share the secret data on this secure channel'
```

**Figure 34: Shows the encrypted and decrypted message using RC6 algorithm**

# References

[1] lea318, “BB84.” May 31, 2021. Accessed: Nov. 28, 2022. [Online]. Available: https://github.com/lea318/BB84

[2] M. Bityutsky, “RC6: RC6 encode-decoder.”.