Report Introduction To Quantum Information And Computation For Robotics

Nicholas Attolino and Antonio Bucciero

January 2024

Contents

Introduction **3**

1 Data Files overview **3**

2 Problem **3**
2.1 Description 3
2.2 Solution 4

3 Results and critical comments **6**

4 Appendix **8**
4.1 Functions used 8

References **9**

Introduction

This work is a report for the assignment of the Introduction To Quantum Information And Computation For Robotics course. The subject of this assignment is a Quantum cryptography protocol, in particular **BB84 Quantum Cryptography Algorithm for Quantum Key Distribution in an hostile environment**. To develop our code we used IBM-Qiskit, an open-source software development kit for quantum programming and computation [[1], [2]].

1 Data Files overview

```
- Assignment_S5656048_S5194849.zip
    - Assignment_S5656048_S5194849
        - CryptoEve.ipynb
        - Report_BB84.pdf
```

As shown, the zip file contains a folder where are present:

- ***CryptoEve.ipynb***, the code file of the entire work;
- ***Report.pdf***, the explanation of the work done.

2 Problem

Two robots or rovers (Alice and Bob, by convention) must exchange secure information in an hostile environment, i.e. the messages exchanged between them might be intercepted. To assure the secrecy of the messages the two robots must encrypt the message with a secure cryptographic key.

2.1 Description

In this work 3 quantum circuits are defined: one for the Sender " *Alice* ", one for the Receiver " *Bob* " and one for the Eavesdropper " *Eve* " to simulate 3 different quantum computers. The objective is to simulate what happens during a communication exchange between 2 quantum computers, especially when a third party tries to intercept their communication to steal data.
It is assumed that *Alice* and *Bob* will use 2 channels for their communication: one public or "unreliable" channel where they will exchange their messages, and one private "safer" one where they will exchange the private quantum key extracted from their previous public communication.
If *Eve* attempts to spy their communication on the public channel, *Alice* and *Bob* should be able to notice her, due to the fact that their keys will differ; in that case, they will proceed to change their public communication channel.

2.2 Solution

The initialization starts with the number of qubits chosen as 8 (knowing that the maximum number allowed is 24, since an error is triggered if more qubits are set).
Then, the quantum circuits for *Alice*, *Bob* and *Eve* are defined and their states and bases are also set (for easier visualization, printed by converting them into strings):

- **States** will be handled as binary arrays of size equal to the number previously set, and filled with random elements. Only *Alice*'s state will be defined a priori because the state is the logic string which, after being encoded in *Alice*'s basis as qubits, will be used as a message.
- **Bases** will be handled as binary array of the same size to be used for the encoding procedures, in which 0 refers to the use of a Computational Basis and 1 to the use of the Hadamard Basis.

The next step is to decide whether an eavesdropper is present on the public communication channel or not. To do this, a flag ***E*** is used to allow testing both when an eavesdropper is present or not.
At this point, *Alice*'s logical state is encoded by converting it into qubits (following *Alice*'s own basis) and assigning them to the quantum state of its corresponding quantum circuit. The function used to do this is called ***encode***. It is a function that will be also used to send qubits between the quantum circuits. Indeed, since qubits are not physically transmitted between quantum computers, it is assumed that the transmission occurs simply in this way:

- Initially, the Sender has the qubits of its own quantum state, encoded in its own base;
- Then, the qubits are in possession of the Receiver, still encoded in the Sender's base;
- Later, the qubits will be processed in the Receiver's base.

If *Eve* is not present:

- Since it was not possible to have access to two quantum computers and an optic fiber to send the qubits representing *Alice*'s quantum state directly to *Bob*, the sending of the message is handled by simply encoding *Alice*'s quantum state onto *Bob*'s quantum circuit (*Bob* simply receives the message encoded in ***encode***;
- Now, the message received is processed by *Bob*, which decrypts it using its own basis through the function ***process***;
- Then, *Bob*'s resulting quantum state is measured in order to use it for computing the encryption key through the function ***measure_state***.

Otherwise:

- Since *Eve* cannot clone nor measure *Alice*'s message and does not know *Alice*'s basis, she will be forced to decrypt the message through a random basis of their own and then send her own resulting quantum state to *Bob*, encoded in her own basis, pretending she didn't disrupt the communication (however, she must take a gamble: her result will be probabilistic, so only if her basis and Alice's basis coincide she will surely be invisible and undetected): this is done through the functions ***encode***, ***process*** and ***measure_state***, used respectively to send *Alice*'s quantum state (encoded in her own base) to *Eve* (as done before for *Bob*), decrypt the qubits using *Eve*'s basis, then measure of *Eve*'s quantum state expressed in her own basis;
- Then, *Eve* will send her quantum state qubits to *Bob*, encoded into *Eve*'s basis, who will process them using his own basis and then measure his own state for computing the encryption key (exactly as done for the *Eve*-less case).

Finally, in both cases, *Alice* and *Bob*'s respective bases are assumed to be shared through their private communication channel, so that they can use them to extract the cryptographic key from their own state through the function ***extract_key***, which also compares them and determines if the communication was compromised by *Eve* (by computing a percentage of similarity between the keys and accepting as valid only communication channels which produce a key with a 90% or higher rate).

3 Results and critical comments

The most significant results obtained during testing are the following:

```
Alice's State:   [1 1 0 1 1 0 1 1]
Alice's Bases:   [1 1 1 1 0 0 1 0]
Bob's Bases:     [1 1 1 0 0 1 1 1]
Bob's Measured State:    [1 1 0 1 1 0 1 1]
Correspondences between the bases:  5
Number of valid qbits for the encription key:  5
Percentage of similarity between the keys:  1.0
Key exchange has been successful!
Alice's Key: 11011
Bob's Key: 11011
```

Figure 1: Without Eve

```
Alice's State:   [0 1 1 0 0 1 1 1]
Alice's Bases:   [0 0 0 0 0 1 1 0]
Bob's Bases:     [0 0 1 1 0 0 0 0]
Eve's Bases:     [0 1 1 1 0 0 1 1]
Eve's Measured State:    [0 1 0 1 0 0 1 1]
Bob's Measured State:    [0 0 0 1 0 0 1 0]
Correspondences between the bases:  4
Number of valid qbits for the encription key:  2
Percentage of similarity between the keys:  0.5
Key exchange has been tampered! Check for eavesdropper or try again
Alice's Key (0101) differs from Bob's Key (0000)
```

Figure 2: With Eve detected

```
Alice's State:   [1 0 1 0 1 1 0 1]
Alice's Bases:   [0 1 1 0 1 0 1 0]
Bob's Bases:     [0 1 1 1 1 1 1 0]
Eve's Bases:     [0 1 1 0 1 0 1 0]
Eve's Measured State:    [1 0 1 0 1 1 0 1]
Bob's Measured State:    [1 0 1 0 1 0 0 1]
Correspondences between the bases:  6
Number of valid qbits for the encription key:  6
Percentage of similarity between the keys:  1.0
Key exchange has been successful!
Alice's Key: 101101
Bob's Key: 101101
```

```
Alice's State:   [1 1 0 1 0 1 1 1]
Alice's Bases:   [1 1 1 0 1 0 1 0]
Bob's Bases:     [0 1 1 1 1 1 1 0]
Eve's Bases:     [1 1 1 0 1 0 0 0]
Eve's Measured State:    [1 1 0 1 0 1 0 1]
Bob's Measured State:    [1 1 0 1 0 1 1 1]
Correspondences between the bases:  5
Number of valid qbits for the encription key:  5
Percentage of similarity between the keys:  1.0
Key exchange has been successful!
Alice's Key: 10011
Bob's Key: 10011
```

Figure 3: Left undetected same bases, right luckily undetected

The figure 1 shows the case in which the public communication channel is not intercepted by an eavesdropper *Eve*: in this case, the key exchange between *Alice* and *Bob* will always be successful (meaning that they will both generate the same key) unless their respective bases are totally different (each couple of single elements of both bases placed in the same position within the base presents a 0 and a 1, never two 0s or two 1s).
In the case shown, *Bob* and *Alice*'s bases are equal in the 1st, 2nd, 3rd, 5th and 7th positions. The logical elements of *Alice*'s state in correspondence of those positions will be 1, 1, 0, 1, 1 respectively, and indeed this sequence will be perfectly matched both in the final measurement of *Bob*'s quantum state and in the generated keys (which will obviously be equal).

By chance, *Bob*'s measured quantum state also perfectly matches Alice's original state. This is due to the fact that the result of *Bob*'s measure on the qubits which are encoded differently in *Alice*'s and Bob's bases is to be intended as probabilistic: since for mismatching elements only one Hadamard Gate is used either by *Alice* or *Bob* (corresponding to the 1s in the bases), the result of the measure on those elements will be either 0 or 1, so it may happen that all elements match, even though some elements of the bases differ (but it isn't always the case, obviously).
Notice that when an Hadamard gate is applied, its result will be probabilistic. If another Hadamard gate is applied on it, the result will be the original qubit (they cancel each other out); so, when an uneven number of Hadamard gates is applied sequentially, the result will still be probabilistic (thus, when measured, it will either give 1 or 0 with a 50% chance). This phenomenon is what may lead to unexpected matches.

The figures 2 and 3 show cases in which the public communication channel is intercepted by an eavesdropper *Eve*. Specifically, the key exchange between *Alice* and *Bob* will be successful only in two cases:

- When *Eve*'s basis and *Alice*'s basis (both chosen at random) match perfectly coincidentally, at least in correspondence of the elements for which *Bob*'s basis also matches with *Alice*'s basis (this is exactly what happens in the left side of figure 3);
- When *Eve* is "lucky", that is when the qubits encoded with a different basis than *Alice*'s one are encrypted with an Hadamard Gate right before *Bob*'s measure; this leads to a probabilistic result of either getting 0 or 1 which may make *Alice*'s and *Bob*'s generated keys identical, similarly to the case previously explained for 1. As a result, *Eve* will be unnoticed, however the higher is the number of qubits in the quantum message, the lower will be the probability of this happening (as shown in the right side of figure 3)).

In all other cases, like in the example shown in 2, the key exchange will not be successful and *Eve* will be detected. This will allow *Alice* and *Bob* to understand that their public channel is not safe. Consequently, they should change their communication channel.

4 Appendix

4.1 Functions used

```
def encode(circuit, state, basis):

    # Sender prepares qubits
    for i in range(len(basis)):
        if state[i] == 1:   # the quantum circuit starts with all ↩
            qubits to 0, so we apply "not" to encode 1
            circuit.x(i)
        if basis[i] == 1:   # if the basis is 1 we encode through ↩
            Hadamard, otherwise we leave it Computational
            circuit.h(i)    # so, when the basis is 0, the qubits are ↩
                encoded with a Computational basis and result
                            # in the same value of the original state, ↩
                                while when the basis is 1 the qubits ↩
                                are proabilistic

    return circuit
```

```
def process(circuit, measurement_basis):

    # Decryption
    for i in range(len(measurement_basis)):
        if measurement_basis[i] == 1:   # when the measurement basis is↩
             1, we decrypt with hadamard, otherwise we leave it as is
            circuit.h(i)

    return circuit
```

```
def measure_state(circuit, num_qubits, name):
    circuit.measure_all() # last thing to be done
    back = BasicAer.get_backend('qasm_simulator')
    key = execute(circuit.reverse_bits(),backend=back,shots=1).result()↩
        .get_counts().most_frequent()
    state = np.zeros(num_qubits, dtype = int)
    for i in range(num_qubits):
        state[i] = key[i]
    print(f"{name}'s Measured State:\t {np.array2string(state)}")

    return state
```

```
def extract_key(state1, state2, basis1, basis2, num_qubits, name1, name2):
    encryption_key1 = ''
    encryption_key2 = ''

    # we use two counters in order to verify the validity and similarity of the two keys
    acc_key = 0
    acc_qbit = 0
    perc = 0
    for i in range(num_qubits):
        if basis1[i] == basis2[i]: # when the bases are equal we extract the qubits to generate the key
            acc_key += 1
            encryption_key1 += str(state1[i])
            encryption_key2 += str(state2[i])
            if str(state1[i]) == str(state2[i]): # when the qubits are also equal the element is valid
                acc_qbit += 1

    print('Correspondences between the bases: ',acc_key)
    print('Number of valid qbits for the encription key: ',acc_qbit)

    if acc_key == 0: # avoids "division by zero" types of errors
        perc = 0
    else:
        perc = acc_qbit/acc_key # if there are more valid matches between the bases than between the qubits themselves, our communication may have been intercepted; we need to define
                                # a safety threshold based on which we accept or discard the generated key
    print('Percentage of similarity between the keys: ', perc)

    if perc < 0.9:           # 90% safety threshold
        print("Key exchange has been tampered! Check for eavesdropper or try again")
        print(f"Alice's Key ({encryption_key1}) differs from Bob's Key ({encryption_key2})")
    else:
        print("Key exchange has been successful!")
        print(f"{name1}'s Key: {encryption_key1}")
        print(f"{name2}'s Key: {encryption_key2}")
```

References

[1] https://www.ibm.com/quantum.

[2] https://www.ibm.com/quantum/qiskit.