Topics of Quantum Computing: Quantum Key Distribution

Rebekka Karrer

July 2020

Abstract

Quantum Key Distribution is one important technique to distribute a private key for Private-Key cryptosystems. It relies on the properties of Quantum Mechanics, eliminating the possibility of cloning or exact measuring of qubits, thus making it impossible for an eavesdropper to obtain information about the key without leaving a trace. The first QKD protocol, the BB84-protocol, realizes this by using non-orthogonal qubits. My implementation of the BB84 protocol with classical postprocessing enables a user to walk through the protocol and observe the effects of the different operations. A description of the code structure and a short tutorial about how to use the prototype can be found after the motivation and the description of the BB84-protocol and the here-implemented classical postprocessing schemes.

1 Motivation

A life without secure encryption – making messaging on the internet, data protection, or online banking impossible – is unimaginable for most people. Every day, everyone will encounter multiple times the problem that they want to send a secret message, like a PIN, to a receiver, for example their bank, without someone else knowing about it. The field of cryptography condensed this situation, which can occur in many facets: Alice, the sender, wants to send Bob, the receiver, a message without Eve, the eavesdropper, knowing the content of the message. To encrypt her messages, Alice can currently choose from two forms of encryption: Public-Key cryptosystems and Private-Key cryptosystems. Public-Key cryptosystems are based on the experience that certain mathematical operations are much faster than the inverse operations, so-called one-way functions. The most famous of these methods is RSA (standing for Rivest–Shamir–Adleman). This cryptosystem is based on the empirical fact that the determination of prime factors grows exponentially with the number whereas the inverse operation, a multiplication of prime factors is computationally cheap. Therefore, the receiver of a message can generate a private key, consisting of two large prime numbers, and a public key, the result of the multiplication of the two prime factors. Bob can then publish the public key. Alice can encode the message with the public key, but only the receiver who possesses the private key can decode the message. In addition to being slow, the major drawback of public key methods is their mathematically insecure basis. It has not been proven that prime factorization is an exponential problem on classical computers; for the time being, this is only an observation about current implementations. Even worse, Peter Shor found an algorithm to solve the problem in polynomial time on a quantum computer. Therefore, traditional Public-Key encryption will not be safe once large quantum computers are available.

The other type of cryptosystems are Private-Key cryptosystems. For this type of encryption, Alice and Bob first share a private key. Then, Alice encodes the message with the private key and Bob decodes this message with the same private key. One approach to do that is the One-Time-Pad which uses a fully random bit string of the length of the message and performs a XOR operation to encode the message. Decoding is also realized by performing a XOR operation. The key challenge of Private-Key cryptosystems is how Alice and Bob can share a private key if they never meet – as it is usually the case with online interactions.

2 Quantum Key Distribution

Quantum Key Distribution is one attempt to solve this problem by using the notion that Eve cannot gain any information without leaving a trace. This is the case because Quantum Key Distribution is, as the name suggests, based on Quantum Mechanics. More precisely, it relies on Heisenberg's Uncertainty Principle and the No-cloning theorem. The first states that by measuring a quantum state one can only obtain incomplete information about a qubit in the general case. The second one excludes the possibility of cloning a quantum state and thereby prevents an eavesdropper from making multiple copies of a state and obtain the information by measuring different copies in different bases. Therefore, the key can be send over a public quantum channel because Alice and Bob will know due to these properties if someone else is eavesdropping.
Since the invention of this field, there have emerged a panoply of protocols. They all principally follow the same work flow. Before starting the Quantum Key Distribution, the two parties, Alice and Bob, have to authenticate themselves to exclude man-in-the-middle attacks where Eve replaces Bob and obtains the key from Alice directly. After this first step, Alice and Bob will exchange some qubits. Because also Alice's and Bob's exchange obeys the law of quantum mechanics, they have to exclude uncertain results. This is done during "Key sifting". After this step, Alice and Bob calculate the quantum error rate to know if an eavesdropper was listening to their exchange. If the error rate is high, they abort and retry in another channel; otherwise, they proceed to classical postprocessing algorithms to first reduce the error by applying error correction and to secondly reduce the information that a potentially undetected Eve has by applying a privacy amplification scheme. In the following, I will explain the BB84 protocol in detail. Further information about Quantum Key Distribution and other protocols can be found in [4], [5], [3], or [2].

3 BB84 protocol

The BB84-protocol is the first published QKD protocol, and it is named after its inventors (Charles H. Bennett and Gilles Brassard) and the year in which it was published (1984). The protocol is proven to be secure if reasonable assumptions are made. In this protocol, each qubit is a classical bit represented in one of two bases. For example, if the z- and the x-basis are used, the four available qubits are the following:

	0	1
+	$\|\psi_{00}\rangle = \|0\rangle$	$\|\psi_{10}\rangle = \|1\rangle$
×	$\|\psi_{01}\rangle = \frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$	$\|\psi_{11}\rangle = \frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$

Table 1: The quantum state given a classical bit and the basis

To start the protocol, Alice randomly chooses one of the four qubits, stores the bit and the basis, and sends the prepared qubit to Bob. Bob randomly chooses one of the two bases, measures the qubit in this basis, and also stores the basis and the resulting classical bit. He then announces to Alice that he has completed his measurement. This is repeated many times until enough qubits are sent for the desired final key length. In the second step, Alice and Bob publicly announce the bases they used to prepare and measure the qubit, respectively. If they used the same basis in an ideal quantum channel without an eavesdropper, it is guaranteed that the bits will be the same too. Constrastingly, if they used different bases, the resulting bits differ with a probability of 50 %, even if they use an ideal channel and no eavesdropper is present. Therefore, both discard all bits where their choice of basis differed. After this step, their keys are the same if the channel is noise-free and no eavesdropper is present. To check whether an eavesdropper is present, they determine their error rate by randomly selecting and comparing a fraction p of their remaining bits. If the error rate is high, they

abort the key exchange and restart it on a different channel because they cannot be certain if an eavesdropper knows parts of their key. If the error rate is low, they continue with classical postprocessing methods. An eavesdropper measuring and resending every single qubit leads to an error rate of 25 %, meaning that an error of approximately 10 % is already considered high.

4 Postprocessing

In realistic scenarios, the detected error rate will be greater than 0 because even if no eavesdropper is present, the channel introduces noise. Therefore, the key exchange is followed by error correction and privacy amplification. As this is not the focus of the topic, we will here use simple schemes to successfully accomplish the task. However, many more advantageous schemes can for example be found in [1].
Error correction can be achieved by Alice selecting two of the remaining bits, announcing their respective positions and the value obtained by performing a XOR-operation. Bob also performs a XOR-operation on the same bits, and announces the result to Alice. If the values coincide, both store the first bit of the two selected bits. If the resulting values differ, the discard both bits. This basic step is then repeated until the whole key is checked and converted in the shorter, error-free key, provided that the initial error rate was low.
Even at this point, a potential eavesdropper could still have partial knowledge about the key. To reduce this knowledge to an arbitrary low amount, privacy amplification is applied. A simple scheme is very similar to error correction: Alice again selects two bits and announces their position, but instead of announcing the XOR-value, Alice and Bob privately calculate the XOR-value and store it instead of the initial two bits. If Eve knows one bit at the beginning, she does not know anything about the final XOR-value. Hence, the knowledge of Eve is effectively reduced at the cost of a shorter final key.

5 Implementation of the BB84 Protocol with GUI

I implemented the BB84-protocol including two simple schemes for postprocessing. It is embedded in a graphical user interface (GUI) to enable the user to see the information that Alice, Bob, and Eve have during each step of the protocol. The user can specify various parameters, and observe how they change the results.

5.1 Code Documentation

The prototype was implemented in Python 3. For the calculations, the libraries numpy and random were used. For the GUI, the built-in Python library Tkinter was used because it is portable and simple to use. The code is structured in six classes: the classes *System* and *Channel*, the base class *Person*, and the derived classes *Alice*, *Eve*, and *Bob*. The interplay of the different classes is displayed in a UML-diagram in Figure 1.
The *System* class contains the handling of the GUI as well as calling the appropriate methods of the *Channel* class to start the next steps. The GUI consists of two main frames: the menu and the process frame. The menu consists of different buttons that lead to different actions. The process frame contains the visual information of the key sharing protocol. I will not explain the methods of the *System* class in detail because – despite being most of the work – for the topic of Quantum Key distribution, the implementation of the other classes is much more relevant. The *Channel* class contains one *Alice*, one *Bob* and – when appropriate – one *Eve*. The channel is the coordinator which connects the information from the different people. Therefore, it has methods for all phases of the protocol. The *Person* class implements the functionality of the people. Most importantly, it has methods for creating and measuring qubits. Each person *Alice*, *Bob*, and *Eve* inherit all methods from the *Person* class. In addition, each person implements a method called *one_step* which implements what one step in the first phase – the transmission and measuring of the qubits – consists of.

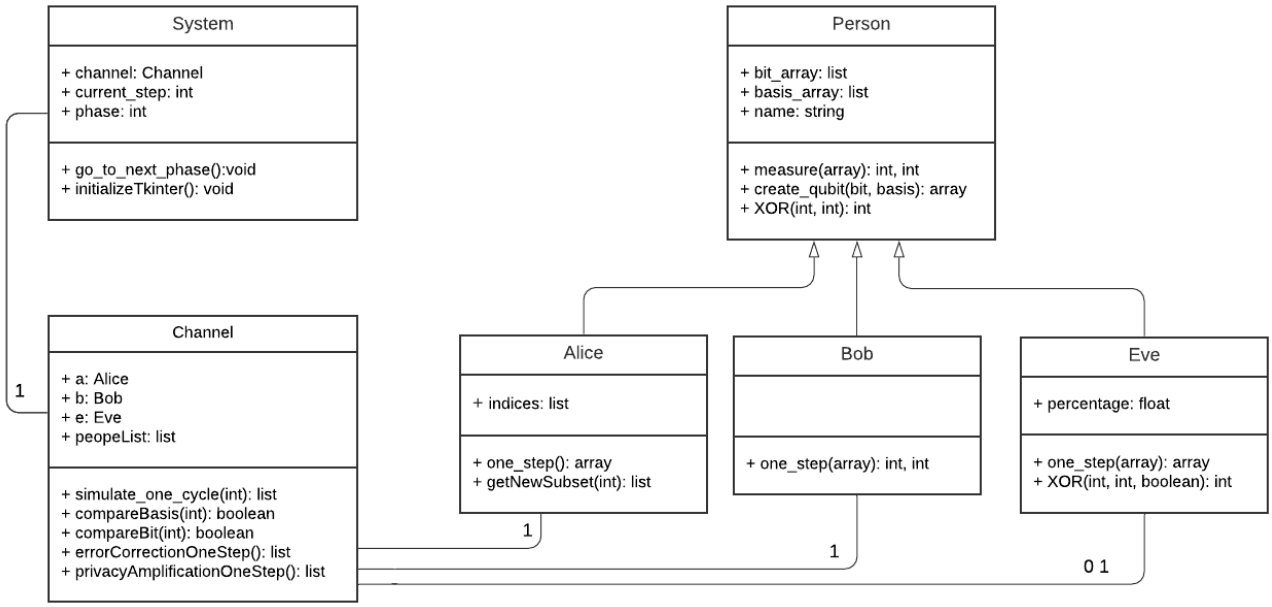

Figure 1: The simulation consists of the six different classes *System*, *Channel*, *Person*, *Alice*, *Bob*, and *Eve*. The system has a channel object, the channel object has an Alice-, a Bob-, and when applicable an Eve-object. All these are derived from the *Person* class.

In the following, I will briefly describe the implementation of the most important features for the BB84-protocol and the postprocessing as detailed above. In this simulation, the qubits are stored and transmitted in the form of density matrices. The form that each density matrix takes can be easily calculated using the Table 1 and the relationship $\rho = |\Psi\rangle\langle\Psi|$. The results can be seen in the following table:

	0	1
+	$\rho_{00} = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$	$\rho_{10} = \left(\begin{smallmatrix} 0 & 0 \\ 0 & 1 \end{smallmatrix}\right)$
×	$\rho_{10} = \frac{1}{2}\left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$	$\rho_{10} = \frac{1}{2}\left(\begin{smallmatrix} 1 & -1 \\ -1 & 1 \end{smallmatrix}\right)$

Preparing a qubit (method *create_qubit*, class *Person*) thus means inputting the desired bit and basis and looking up and returning the corresponding density matrix. Alternatively, the bit and basis can also be randomly chosen in the function itself. A qubit is measured (method *measure*, class *Person*) by determining the result of the measurement operator in the right basis:

$$\langle A \rangle = \mathrm{tr}(A\rho) \tag{1}$$

A is in the z- and x-basis

$$A_z = \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \qquad A_x = \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2}$$

respectively. The evaluation of equation 1 leads to a value between -1 and 1 (-1 corresponds to a classical bit of 1, 1 corresponds to a classical bit of 0, 0 corresponds to a totally mixed state). To determine the outcome of a measurement, a random number between -1 and 1 is generated, and if the random number is greater than the value obtained from the measurement, the result of the measurement is 1. If the random number is smaller than or equal to the value, the result of the measurement is 0. As the result of the calculation of 1 is 0 in case of non-orthogonal basis and qubit, the resulting bit is chosen randomly (not accounting for

possibly small numerical rounding errors). One cycle of creating and measuring a qubit consists of two or three steps (method *simulate_one_cycle*, class *Channel*). First Alice prepares a qubit, then if an eavesdropper exists, the eavesdropper will draw a random number between 0 and 1. If the number is smaller than the personal eavesdropping percentage, the eavesdropper will perform an intercept-resend attack by measuring the qubit and preparing another qubit according to the measured information. Finally, Bob will measure the qubit. All people store the information about the qubit in their attributes *bit_array* and *basis_array*.
The functionality of next steps is implemented straightforward for Alice and Bob. The key sifting (method *compareBasis*, class *Channel*) compares the bases of Alice and Bob at each index and they keep only the bits where the bases were the same. Eve only stores the bits where her basis agrees with Alice's and Bob's bases. For the calculation of the error rate, Alice first selects a random subset of the indices of size specified by the user (method *getNewSubset*, class *Alice*). Then the bits at these indices are compared, the error rate is calculated, and all people discard the bits at these positions. During error correction (method *errorCorrectionOneStep*, class *Channel*), Alice selects two bits, Alice and Bob announce their XOR-value. If they coincide, they keep the first bit, if they differ, they discard both bits. If they store the first bit and Eve also knows the one of the two selected bits, she can store the right bit. If she knows the first bit, she stores the first bit. If she knows the second bit, she performs an XOR-operation on the second bit and the XOR-value of Alice and Bob. She stores the result of her XOR-operation, which equals the first bit that Alice and Bob stored. In privacy amplification (method *privacyAmplificationOneStep*, class *Channel*), Alice chooses two indices and Alice and Bob replace the values at the two indices by an XOR-operation. If Eve knows both values, she also stores the result of the XOR-operation. If not, she does not know the number.

5.2 Tutorial

In this section, I will provide a guideline of how my simulation of the BB84-protocol including the two post-processing schemes can be used. The implementation can be obtained by cloning the git repository found at `https://github.com/rebekkaka/BB84-protocol`. Then the simulation is started by running the file 'executable.py' with the command 'python3 executable.py' on the console inside the folder 'BB84-protocol'.
Then a Tkinter window appears (see Figure 2(a)) and the user has to enter a number between 0 and 1 to specify the average eavesdropping rate. If 0 is entered, no eavesdropper will be present. By clicking on 'Start', the first phase 'Transmission of qubits' appears (see Figure 2(b)). The user can choose his or her actions at the left side and sees the results of those at the right side. By clicking on 'Simulate one cylce', one cycle is simulated. This means that the bit and basis chosen by Alice as well as the resulting schematic of a qubit, the chosen basis and the measured bit by Bob, and when Eve intercepted also her choice of basis, her measured bit, and the resulting qubit schematic are displayed on the right. By entering a positive integer in the entry and clicking on 'Simulate multiple cycles', multiple (the number that was specified) cycles are displayed at once. Once one has simulated enough qubit transmission, one can click on 'Go to next phase'. It is recommended to simulate more than 70 cycles to have enough qubits for all phases.
The next phase is 'Key sifting' (see Figure 2(c)). By clicking on the button 'Compare bases', the bits and bases where the bases were the same are highlighted in green for Alice and Bob and in light green for Eve. The resulting shorter key is displayed below. In the next phase, 'Error rate' (see Figure 2(d)), the user has to first specify the desired number of samples that should be used for the calculation of the error rate. By then clicking on 'Compute error rate', the specified number of samples is selected at random and highlighted in orange. The error rate is displayed and the user has to choose between aborting the key distribution or continuing by clicking on the button 'Abort' or 'Continue with postprocessing', respectively.
If 'Abort' is chosen, the simulation is restarted. If 'Continue with Postprocessing' is chosen, the renewed key without the bits used for comparison is displayed and the user has the choice between 'One error correction step' or 'All error correction at once' (see Figure 3(a)). By clicking on 'One error correction step', the two

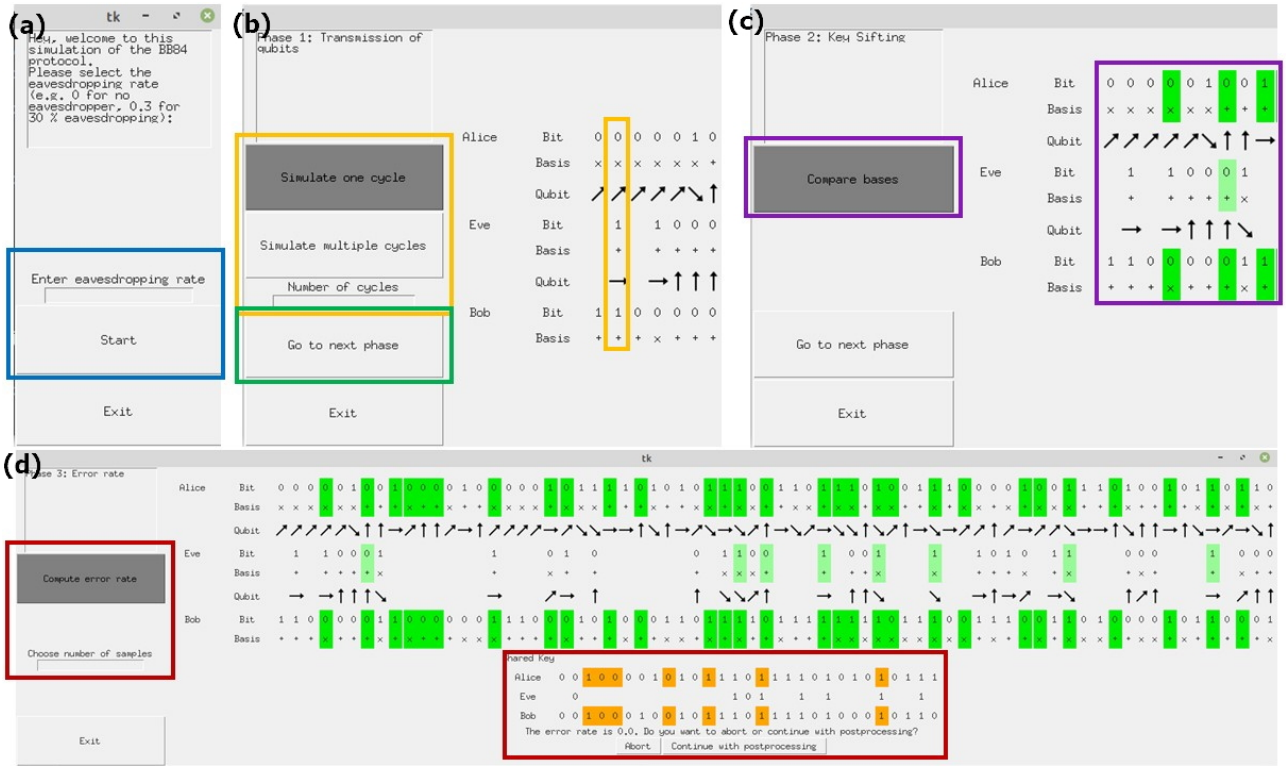

Figure 2: (a) The simulation is started by entering a number between 0 and 1 to specify the eavesdropping rate and then clicking on start (blue box). (b) One can choose to simulate one cycle or enter a number and simulate multiple cycles at once. Each cycle is depicted as one column on the right. After one phase is finished (in phase 1 once the user decides to be finished), one can click on 'Go to next phase' (green box) to go to the next phase. (c) By clicking on 'Compare bases', the information using the same bases is highlighted in green for Alice and Bob or in light green for Eve (purple box). (d) To compute the error rate, one enters a number of samples smaller than the total number of remaining samples and clicks on 'Compute error rate'. The selected subset is highlighted in orange, the calculated error rate is printed below and the user is given the choice to 'Abort' or to 'Continue with Postprocessing' (red boxes).

chosen bits are highlighted in orange. The first bit, which might be stored, is colored in slightly darker orange. The resulting, hopefully error-free key is displayed below the current version of the key. Previously selected bits cannot be selected again and are colored in yellow. By clicking on 'All error correction at once', all the remaining error correction steps are performed at once. The last phase is 'Privacy Amplification' (see Figure 3(b)). By clicking on 'One PA step', the two randomly selected bits are highlighted in orange and the resulting final, shorter key is displayed below. Previous bits cannot be reselected and are highlighted in yellow. By clicking on 'All PA steps', all remaining privacy amplification steps are performed at once. Once all steps are completed, a final message is displayed, and the user can choose to 'Restart' or to 'Exit' (see Figure 3(c)).

5.3 Expected Learnings

A few observations are likely to be made using this simulation of the BB84-protocol. First, one can familiarize him- or herself with the properties of quantum measurements, namely that measurements in the same basis always lead to the same results and measurements in non-orthogonal basis lead to random results. Second, it is apparent that a large amount of qubits is necessary to generate a reasonably long key, for example for a One-Time-Pad, which requires a key of equal length as the message in bits. Third, the statistics for the error

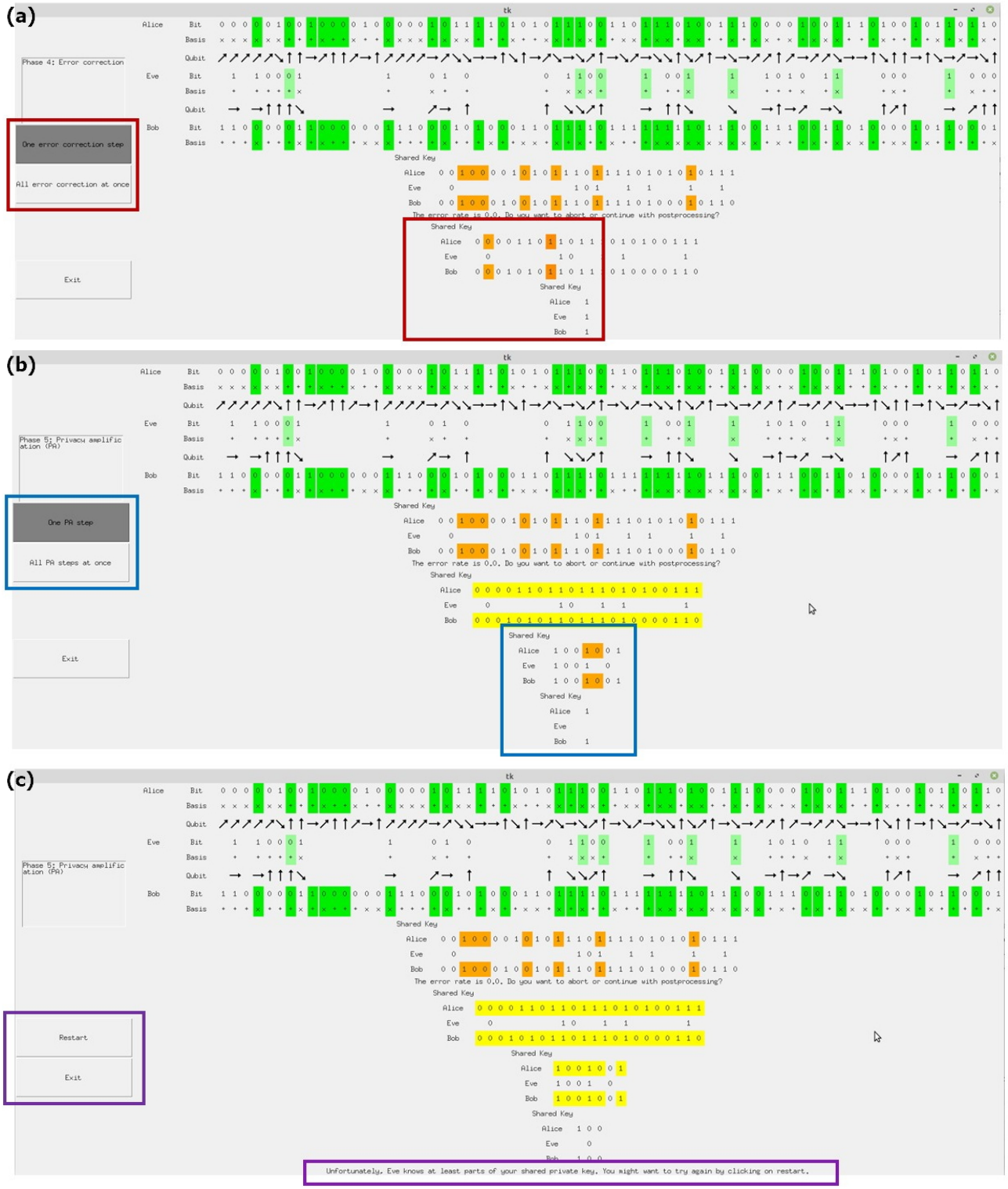

Figure 3: (a) One error correction step is done by clicking on 'One error correction step'. This leads to a highlighting of the two selected bits, the first one in slightly darker orange than the first one. If the XOR-values are the same, the first bit is stored and displayed in the updated key below. Previously processed bits are highlighted in yellow. One can speed up the process by clicking on 'All error correction at once'. (b) One privacy amplification step works very similar. The randomly selected bits are highlighted when clicking on 'One PA step', the result is displayed below. The process is faster when clicking on 'All PA steps'. (c) After the protocol is finished, a custom message is printed, and the user can restart or exit by clicking on the appropriate button on the left.

rate in this small scale simulation are bad because only a (too) few bits are taken into consideration for the calculation. This can also be seen in the tutorial example in Figure 2 and 3. Altogether, my simulation of the BB84-protocol with postprocessing provides an interactive way to familiarize oneself with the BB84-protocol.

References

[1] Gilles Brassard and Louis Salvail. "Secret-Key Reconciliation by Public Discussion". In: (1994), pp. 410–423.

[2] Ivan B. Djordjevic. *Physical-Layer Security and Quantum Key Distribution*. Springer, 2019. ISBN: 9783030275655.

[3] N. et al. Gisin. "Quantum Cryptography". In: *Reviews of Modern Physics* 74.1 (2002). DOI: 10.1103/RevModPhys.74.145.

[4] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010. ISBN: 9781107002173.

[5] Xiaoqing Tan. "Introduction to Quantum Cryptography". In: *IntechOpen* (2013). DOI: /10.5772/56092.