Amsterdam University College

Bachelor's Thesis

Implementing Quantum-Cryptographic Protocols using SimulaQron

A quantum-internet simulation of 1-2 oblivious transfer in the noisy-storage model

Author
Lynn Engelberts
Major in Sciences
Amsterdam University College
lynn.engelberts@student.auc.nl

Tutor
Michael P. McAssey
Amsterdam University College
m.p.mcassey@auc.nl

Supervisor
Christian Schaffner
QuSoft
University of Amsterdam
c.schaffner@uva.nl

Reader
Michael Walter
QuSoft
University of Amsterdam
m.walter@uva.nl

June 3, 2019

Abstract

Research is currently devoted to the development of a large-scale quantum network—also known as a *quantum internet*. Since a quantum internet is expected to be realised in the near future, a quantum-internet simulator—called SimulaQron—has been created to function as framework for the development of quantum-internet applications [DW18]. One of the expected applications of a quantum internet is the cryptographic primitive secure two-party computation. However, the suitability of SimulaQron for cryptographic applications has not yet been examined. In particular, this thesis demonstrates the potential and limitations of SimulaQron as environment for implementing and analysing cryptographic protocols. Furthermore, we evaluate whether the considered protocols are suitable for quantum-internet applications. To the best of our knowledge, this thesis provides the first implementations of quantum-cryptographic protocols in SimulaQron. Our implementations are based on the protocols for 1-2 oblivious transfer (1-2 OT) proposed in [Sch10]. 1-2 OT is a simple two-party primitive on which any protocol for secure two-party computation can be based [GV88; Kil88]. Assuming that the laws of quantum mechanics hold, the protocols in [Sch10] are shown to be secure against attacks in the noisy-storage model—a realistic cryptographic model which assumes that the adversary's quantum storage is noisy. Our work is based on a critical literature analysis of 1-2 OT in the noisy-storage model and quantum internet, followed by an implementation of the protocols provided in [Sch10] using SimulaQron. Our implementations will allow us to evaluate whether these protocols are easy to implement and suitable for quantum internet. By analysing 1-2 OT in the noisy-storage model as well as evaluating the cryptographic applications of a quantum internet and the potential of SimulaQron, this research will thus contribute to both the field of quantum cryptography and the area of quantum-internet development.

Keywords

1-2 oblivious transfer (1-2 OT); error correction; noisy-storage model; quantum internet; SimulaQron

Acknowledgements

Firstly, I would like to thank my supervisor Christian Schaffner for his patience and guidance during this project and for his time to meet on a regular basis. By introducing me to the area of quantum cryptography and information theory, Christian provided me with the background and notions necessary for this Bachelor's thesis. My grateful thanks are also extended to Axel Dahlberg for taking the time and effort to help me with any questions related to SimulaQron. Finally, I would like to thank Michael Walter for taking the time to read my work.

Contents

1 Introduction

By 2020, the Dutch research centre QuTech aims to have built a quantum network between four cities in the Netherlands (Amsterdam, Delft, Leiden, and The Hague) [ALA18; DW18; QuT]. The ultimate goal is to construct a large-scale quantum network so that quantum information can be exchanged between remote quantum processors over an arbitrarily long distance. Such a large-scale quantum communication network is also known as a *quantum internet* and is expected to enable quantum computation and quantum communication over arbitrarily long distances [Cas18; DW18; DLH17]. Despite the fact that many possible applications of a quantum internet have been suggested in research (e.g. see [Weh08]), the real potential of a quantum internet is unknown until it will in fact be realised. Nevertheless, the presumed arrival of a quantum internet will require software that can be run over the quantum network. Researchers have, therefore, created a quantum internet simulator—called SimulaQron—with the purpose to function as a test-bed for writing software for quantum internet applications [ALA18; DW18].

One of the potential applications of a future quantum internet is two-party cryptography [DW18; Weh08]. Cryptography is concerned with the tasks of achieving communication or computation between two or more parties, even if the parties do not trust each other [NC10]. A typical example is that two parties want to communicate securely over a communication channel. We often refer to these two parties as "Alice" and "Bob", and to a (third) party attempting to intercept their messages as the adversary or attacker. A cryptographic protocol then describes how Alice and Bob can perform this task securely. In classical cryptography, the security of such cryptographic protocols is often based on the hardness of an unproven mathematical conjecture and on the assumption that the attacker has limited computational power [DLH17; KWW12]. However, the security of these protocols could thus be affected if the attacker obtains more (e.g. quantum) computational power. For instance, the algorithm developed by Shor [Sho95] for integer factorisation on a quantum computer could break the widely-used public-key cryptoscheme called RSA [RSA78]. It is therefore desirable to aim for protocols that are information-theoretically secure—that is, the security of the protocol can be proven without restrictions on the attacker's computational power [Sca+09].

Whereas, on the one hand, the laws of quantum mechanics allow for breaking currently used classical cryptoschemes, these laws could, on the other hand, be exploited to design quantum protocols for cryptographic tasks, which is the topic of quantum cryptography. In fact, information-theoretical security can be achieved in the case of quantum key distribution (QKD)[1], which enables two parties to establish a shared secret key. Nevertheless, not all cryptographic tasks can be performed with information-theoretical security, even with the help of quantum mechanics. This limitation holds, in particular, for secure two-party computation, which involves cryptographic tasks in which two parties want to exchange information but do not trust each other.[2] More precisely, it has been shown that additional assumptions on the attacker are inevitable in order to allow for secure two-party computation [BCS12; Lo97; LC97; May97].

Consequently, research has been devoted to designing cryptographic models for secure two-party computation protocols which introduce such additional assumptions on the adversary. Two cryptographic models that have been proposed are based on *physical* (instead of computational) assumptions, mimicking the technical difficulties that one encounters when storing quantum information: the bounded-quantum-storage model [Dam+05; Dam+07] and the noisy-

[1] For example, see [TL17] for a security analysis for QKD protocols.

[2] Note that, contrary to the previous example and QKD, Alice and Bob now do not trust each other, and thus the "adversary" can be either of the two. A concrete application of secure two-party computation is the so-called millionaire's problem in which Alice and Bob want to find out who is the richest of the two without revealing to the other how much money he or she has [Yao82].

storage model [KWW12; Sch10; STW09; Weh08]. Whereas the first model gives an explicit upper bound on the adversary's quantum storage during the protocol, the latter assumes that the quantum storage is noisy [KWW12; Sch10; STW09; Weh08]. The noisy-storage model is perceived as more realistic compared to the bounded-quantum-storage model, as it reflects more accurately the prevailing technical difficulties of storing quantum information [KWW12; Sch10].

Several protocols for secure two-party computation have been designed for the noisy-storage model, among which protocols for the primitive called *1-2 oblivious transfer* (1-2 OT), such as provided in [KWW12] or [Sch10]. 1-2 OT is particularly interesting, since it has been shown that any protocol for secure two-party computation can be based on this primitive [GV88; Kil88]. 1-2 OT is a cryptographic primitive in which Alice has two secret strings (messages) m_0, m_1 and Bob has a bit $c \in \{0, 1\}$ and the goal is that Bob receives m_c corresponding to his choice bit c, but neither Alice learns Bob's choice c nor Bob learns the remaining string of Alice [GV88; KWW12]. The protocols for 1-2 OT in the noisy-storage model require Alice and Bob to be connected by a quantum channel. Since it is realistic to assume that in any practical implementation of 1-2 OT such a quantum channel may be unreliable and hence induce errors on the transmitted quantum information, protocols have also been suggested that are *robust* against errors caused by the quantum channel, such as the one in [Sch10]. In particular, both the idealised and the robust protocol provided in [Sch10] are shown to be information-theoretically secure against any noisy-storage attack, as long as the laws of quantum mechanics hold and the amount of noise in the quantum storage channel is above a particular level.

Since secure two-party computation is one of the expected applications of a future quantum network [Weh08] and any secure two-party computation primitive can be based on 1-2 OT, it is significant to evaluate the implementation of a protocol for 1-2 OT over such a quantum network. In particular, since SimulaQron has been designed to develop software for future quantum-internet applications, it is important to verify its suitability for implementing cryptographic tasks. Nonetheless, there are to the best of our knowledge no implementations of any cryptographic protocols using SimulaQron. Therefore, the present paper aims to implement both the idealised and the robust protocol for 1-2 OT from [Sch10] in SimulaQron in order to (i) examine whether the selected protocols are adequate for quantum internet applications, and (ii) evaluate the extent to which SimulaQron is a suitable environment for implementing and analysing cryptographic protocols.

Methodology

The present research consists of a critical and extended literature review, followed by an implementation of the two protocols from [Sch10] using SimulaQron [Sim].

By critically analysing the existing literature on 1-2 OT in the noisy-storage model, this paper attempts to justify the choice for the protocols in [Sch10] and to be able to provide a deeper understanding of the protocol for implementation using SimulaQron. Moreover, in order to clarify the extra steps for error-correction for the implementation of the robust protocol in an unreliable quantum channel, it was necessary to study the error-correction techniques that are available. Furthermore, an analysis of the existing literature on quantum internet and SimulaQron was used for understanding the working of SimulaQron and to evaluate the potential of SimulaQron.

SimulaQron has been chosen as environment for our implementations, since SimulaQron allows for simulating the quantum network between Alice and Bob and hence enables to perform (simulated) quantum communication, which is required for implementing the protocols. In order to carry out the implementations, we have used several tools provided by the Python classical-

quantum-combiner (CQC) library that comes with SimulaQron and made the implementations feasible. We have written our code in Python [Pyt] and hence used several tools provided by Python and its libraries. For the decoding part, we use the source code from [Fil], which is also written in Python.

Knowledge utilisation

This research provides new insights into the area of quantum internet development. Especially since a quantum internet is not yet realised, the possible applications of a future quantum internet are still speculative. Therefore, by implementing and evaluating one of the presumed quantum-internet applications—i.e. cryptography—in a simulated quantum network, our research contributes to a deeper understanding of the suitability of a quantum internet for such cryptographic applications.

Moreover, since there are to the best of our knowledge no previous implementations of cryptographic applications using the recently developed SimulaQron, the results of this research explain the potential of SimulaQron for developing such applications. In addition, we shed light on the possibilities and current limitations of SimulaQron in general, which is of use for current and future users of SimulaQron and may contribute to a further development of this quantum-internet simulator. In fact, we may in this way also contribute to the development of the quantum network in the Netherlands that is planned to be realised in 2020, since a version of the CQC interface[3] used by SimulaQron is intended to be available on this quantum network as well.

Lastly, this research contributes to the field of quantum cryptography as there are to the best of our knowledge no quantum-network implementations of the protocols in [Sch10]. The only implementations of these and similar 1-2 OT protocols for the noisy-storage model are achieved by using hardware for QKD, such as done in [Erv+14]. Furthermore, since our code is written in the high-level language Python and publicly available on GitHub, our code can be used to provide an illustration of the protocols for 1-2 OT for learning objectives.

Outline of the paper

The remaining part of the paper is organised in the following way. Section 2 describes some terminology of quantum mechanics used in this paper. Section 3 provides an analysis of the noisy-storage model and a description of the protocol for 1-2 OT in the idealised setting. Section 4 addresses how to correct errors in the setting of an unreliable quantum channel and explains the resulting robust protocol for 1-2 OT. Section 5 presents a background on quantum internet and SimulaQron. The results of our implementations are provided in Section 6, followed by a discussion. Finally, we conclude our work and provide some directions for future research in Section 7.

[3] See Section 5.

2 Background to quantum theory

In this section, we provide a brief introduction to quantum mechanics and focus on the concepts that are relevant for the remaining part of this thesis. We assume that the reader is familiar with the basic notions of linear algebra. We use the definitions provided in [NC10].

Qubits

Recall that a classical binary bit either has value 0 or 1. The quantum version of the bit is often called a *qubit*, short for quantum bit. Similar to a classical bit, a qubit can be in state $|0\rangle$ or $|1\rangle$, where $|\rangle$ is the frequently-used *Dirac* notation. However, the main difference is that a qubit can actually be in any linear combination of these two states.

More precisely, we define a qubit as a vector in a two-dimensional complex vector space $\mathbb{C}^2$ with inner product, also called a Hilbert space. The states

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \text{and} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

form an orthonormal basis for the Hilbert space, often referred to as the *computational basis*. We write the most general quantum state as: $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$, where $\alpha_0, \alpha_1 \in \mathbb{C}$ are normalised so that $|\alpha_0|^2 + |\alpha_1|^2 = 1$. Such a state is often called a *superposition* of states $|0\rangle$ and $|1\rangle$. Moreover, we call α_i the amplitude of state $|i\rangle$ for $i \in \{0, 1\}$.

Measurement postulate

Qubits act in a probabilistic manner when measured. According to the Copenhagen interpretation of quantum mechanics, a measurement on a quantum state irrevocably changes its state: after a measurement on an arbitrary quantum state $|\psi\rangle$, if the outcome is i, then $|\psi\rangle$ is *collapsed* (i.e. changed) to state $|i\rangle$. Clearly, if the qubit is in state $|\psi\rangle = |0\rangle$ (i.e. $\alpha_0 = 1$, $\alpha_1 = 0$), then we get outcome 0 with probability 1. However, it may seem less trivial if a qubit is in superposition: if the qubit is in state $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$ with α_0 and α_1 both nonzero, we either get outcome 0 with probability $|\alpha_0|^2$ or 1 with probability $|\alpha_1|^2$.

BB84 states

The following four states are often referred to as the BB84 states, since these states were used in the BB84 protocol [BB84] for QKD (named after the authors Charles Bennett and Gilles Brassard and the year in which it was proposed).

$$|0\rangle \qquad |1\rangle \qquad |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \qquad |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

As aforementioned, $\{|0\rangle, |1\rangle\}$ is called the computational basis. We will refer to $\{|+\rangle, |-\rangle\}$ as the *Hadamard basis*.

Operators

An operator acts on a quantum state and enables us to change the state of the corresponding qubit. An operator is defined as a linear transformation on the Hilbert space $\mathbb{C}^2$. By the laws of quantum mechanics, an operator U must be unitary, i.e. $U^\dagger U = I$, where I represents the identity, and $U^\dagger$ means the conjugate transpose of U. Below, we provide two examples of operators, which will also be used in the implementations of this paper.

The first operator we consider is denoted X. It acts as follows on the BB84 states: $X|0\rangle = |1\rangle$, $X|1\rangle = |0\rangle$, $X|+\rangle = |+\rangle$, and $X|-\rangle = -|-\rangle$. In other words, if the qubit is in one of the computational basis states, the X operator swaps the states. Note the connection to the logical `NOT` gate.

The second operator we consider is the *Hadamard* operator H, which acts on the four BB84 states as follows: $H|0\rangle = |+\rangle$, $H|1\rangle = |-\rangle$, $H|+\rangle = |0\rangle$, and $H|-\rangle = |1\rangle$.

An operator can be represented by a matrix. The matrices corresponding to X and H are:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

3 1-2 oblivious transfer in the noisy-storage model

As aforementioned, 1-2 oblivious transfer (1-2 OT) is a cryptographic task belonging to the field of secure two-party computation on which any other protocol for secure two-party computation can be based [GV88; Kil88]. Oblivious transfer was first introduced by Rabin [Rab81] and has been generalised to 1-2 OT by Even, Goldreich, and Lempel [EGL82]. We recall that in the setting of 1-2 OT, Alice has two secret strings m_0, m_1 and Bob chooses a bit $c \in \{0, 1\}$. The task that 1-2 OT needs to accomplish is that Bob receives Alice's input string m_c corresponding to his choice bit c, without revealing c to Alice and without learning the other string of Alice. Note that here, and in the remaining part of this thesis, "Alice" and "Bob" refer to the two parties that want to achieve the task of 1-2 OT, and that the "attacker" can be any of these two parties.

3.1 Additional assumptions

We explained in Section 1 that even with the possibility of quantum communication, protocols for secure two-party computation—and hence 1-2 OT—must impose additional assumptions on the adversary in order to achieve a desired level of security [BCS12; Lo97; LC97; May97]. It is reasonable to aim for protocols for secure two-party computation that are based on feasible and realistic assumptions. Firstly, feasibility is important to ensure that the protocol can be used in practice. Moreover, the assumptions should be realistic in the sense that they take into account the current state of the art and technological limitations [KWW12; Sch10]. Therefore, two cryptographic models have been proposed that are based on *physical* (instead of computational) assumptions: the bounded-quantum-storage model [Dam+05; Dam+07] and the noisy-storage model [WST08; STW09; Sch10; KWW12].

Bounded-quantum-storage model

The physical assumption in the bounded-quantum-storage model is that the adversary's quantum memory size is limited to a certain number of qubits. No restriction is imposed on the adversary's classical memory or computing power. In this model, the version of OT that was introduced in [Rab81] is proven to be information-theoretically secure [Dam+05]. In particular, if Alice and Bob are honest, they do not need any quantum memory and the protocols can only be broken if the dishonest party has a quantum memory of size at least $\frac{n}{2}$, where n is the number of qubits transmitted during the protocol [Dam+05]. Moreover, the findings of [Dam+05] have been extended to the case of 1-2 OT: 1-2 OT is shown to be secure against a dishonest player whose quantum memory is of size at most $\frac{n}{4} - 2l$ [Dam+07]. Here, n amounts again to the number of qubits that are transmitted during the protocol, and ℓ (which may be a fraction of n) is the length of Alice's input strings m_0, m_1. In addition, [Dam+05; Dam+07] point out that their protocols also provide security in the case of imperfect quantum communication, a situation that will be covered in Section 4.

There are three reasons why we may consider the bounded-quantum-storage model. Firstly, if Alice and Bob merely communicate over a classical channel—and thus not use quantum communication—secure 1-2 OT can only be achieved if the adversary's classical memory size is at most quadratic in memory size of an honest party [DM04]. However, using quantum communication, the protocols for the bounded-quantum-storage model only require a bound on the size of the adversary's *quantum* memory. In particular, these protocols achieve a desired level of security regardless of the size of the adversary's classical memory, which, consequently, allows for a bigger ratio between the memory size of the honest parties and that of an adversary than would be possible in the classical setting. In fact, the honest parties do not need any

quantum storage at all, and the bound on the adversary's quantum memory size only depends on the number n of qubits that have been sent during the protocol. Therefore, the bounded-quantum-storage model allows us to outdo the optimal bound for which security is obtained in the case that a dishonest player only has access to a *classical* memory [Dam+05]. Secondly, the limited size of the adversary's quantum memory ensures that a dishonest party will lose information: being unable to store all qubits, the adversary must store some information in his classical memory (i.e. by measuring the qubits) [Dam+05]. It is, in fact, this irreversible loss of information what allows for 1-2 OT in the bounded-quantum-storage model [Dam+05]. The third reason to consider the bounded-quantum-storage model is a practical one: whereas quantum transmission and quantum measurements—needed for the honest parties—is possible with the present state of the art, storing only "a single qubit for more than a fraction of a second" is technologically still very difficult [Dam+05], impeding the storing possibilities of a dishonest party.

Despite the aforementioned reasons, it is questionable whether any practical situation will impose such a limit on the quantum memory size as the bounded-quantum-storage model requires [WST08]. In fact, [Dam+05] argue that it is more realistic to consider the case that the adversary's quantum memory is noisy, instead of bounded. Therefore, we describe the resulting model and its advantages below.

Noisy-storage model

Whereas the bounded-quantum-storage model gives an explicit upper bound on the adversary's quantum memory size during the protocol, the noisy-storage model[4] assumes that the quantum storage (i.e. the quantum memory) is noisy [KWW12; Sch10; STW09; WST08]. More precisely, this model allows an adversary to have the best available quantum memory, but this memory is affected by noise over time: the longer the adversary stores quantum information, the higher the amount of prevailing noise [Sch10]. Two-party protocols for the noisy-storage model (such as the protocols considered in this paper) make use of this time-dependency by inducing a waiting time for the participating parties during the protocol: after this waiting time, the higher level of noise in the adversary's quantum memory results in information loss, see Section 3.2. As a consequence, we can evaluate the security of a protocol for this model in terms of the noise level [WST08]. Furthermore, note that the bounded-quantum-storage model can, in fact, be obtained from the noisy-storage model by simply assuming that there is no noise in the quantum memory but that the input of the quantum memory (i.e. the number of qubits that can be stored) is limited [KWW12].

By exploiting the technological difficulty of building a quantum computer—i.e. the presence of noise—the noisy-storage model has a significant practical value. In fact, the noisy-storage model reflects the prevailing technical difficulties of storing quantum information more accurately than the bounded-quantum-storage model. First of all, current quantum memories are not completely reliable. In particular, with current technology, the time during which quantum information can be stored without going completely lost is limited (e.g. see [Zho+12]). The imperfection of current quantum memories results from the difficulty to keep the stored quantum information unaltered over time and the problem that quantum information (i.e. the photons in which the quantum information is encoded) goes lost during storage [Erv+14]. In fact, [KWW12] argues that it is not even known whether it is physically possible to build a fully reliable memory in the future. In addition, noise may arise when quantum information

[4]Since the noise occurs in the *quantum* memory, researchers sometimes also call this model the *noisy-quantum-storage* model. However, in this thesis we stick to the more common term *noisy-storage* model. In particular, we assume in this thesis that there is no noise in the classical communication channel and classical storage at all.

is transferred onto a different physical carrier representing the quantum memory; for example, when photonic qubits are transferred onto an atomic ensemble or an atomic state [STW09]. In fact, this transfer onto the quantum memory is often already noisy [KWW12], irrespective of the (im)perfection of the quantum memory itself [Weh+10].

Yet, we should note that security is only guaranteed if the aforementioned technological challenges prevail [Erv+14]. Nevertheless, the noisy-storage model is resistant against technological improvements in the future (i.e. security can still be obtained) as long as the quantum memory size has a finite upper bound [Erv+14]. This resistance of the model results from the fact that the level of security depends on the number of qubits that are transmitted during the protocol, and hence can be monitored. In other words, if the dishonest party has access to a larger quantum storage, the required level of security can still be obtained by sending more (quantum) information during the protocol [Erv+14]. Hence, as long as an adversary does not have access to an unlimited and noiseless quantum memory in the future—which can be perceived a reasonable assumption—we conclude that the noisy-storage model is a realistic and feasible model.

3.2 Security in the noisy-storage model

Given the aforementioned reasons why the noisy-storage model could be seen as a realistic reflection of the prevailing technical difficulties, we based our implementations on protocols for 1-2 OT in the noisy-storage model.[5] As previously explained, the noisy-storage model allows a dishonest party to have access to an unlimited quantum memory, yet this quantum memory is noisy. This section provides an intuitive description what such a noisy quantum memory entails for a dishonest party and why this may allow us to achieve security. We also describe another tool that is used in the considered protocols to provide security in the noisy-storage model, called *privacy amplification*.

Noisy-quantum-storage attacks

First of all, note that in the considered protocols for 1-2 OT in the noisy-storage model only a dishonest Bob (i.e. the receiver) can take advantage of his quantum memory, since he is the only one that receives quantum information. In particular, security against a dishonest Alice does not depend on her quantum memory because the only information she receives from Bob is classical information. Therefore, we only consider noisy-quantum-storage attacks by a dishonest Bob. Suppose now that Bob tries to store an incoming quantum state in his quantum memory. Since the quantum memory is noisy, the state is affected by noise and *decoheres* over time [KWW12], i.e. the quantum system interacts with its environment and information goes lost. Then, security in this model can be achieved by imposing a waiting time during the protocol: the quantum information that a dishonest Bob may store in his quantum memory, undergoes noise during this waiting time, which results in loss of information. Hence, in this way, we may exploit the noise in the quantum memory in order to achieve the desired level of security. Nevertheless, we emphasise that a dishonest Bob is still allowed to perform any encoding attack on the received information (i.e. the attacker has unlimited computational power and can do perfect quantum operations) and has an unlimited noiseless classical storage. Also, after the waiting time, Bob can do any decoding attack. See Figure 1 for an illustration of a noisy-quantum-storage attack by dishonest Bob.

[5] For a formal definition of the noisy-storage model, we refer the reader to [Weh+10].

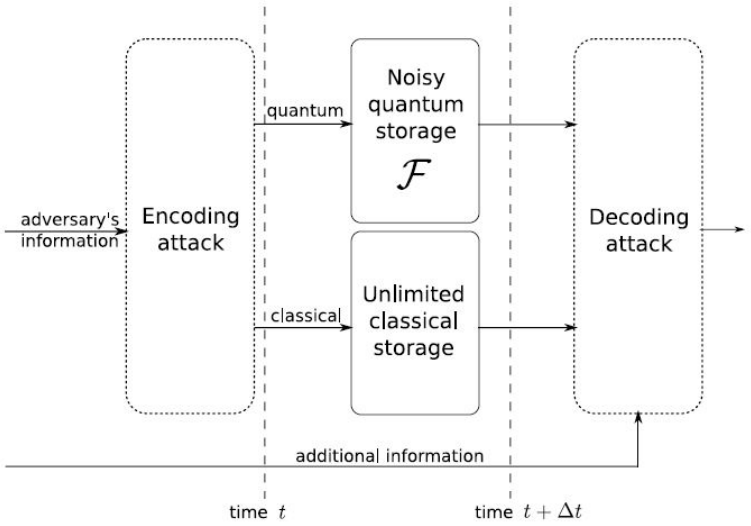


Figure 1: (Source: [Weh+10]) An illustration of an attack by dishonest Bob. Here, $\mathcal{F}$ represents the noisy-quantum memory (as formally defined in [Weh+10]) and Δt the waiting time during the protocol.

Privacy amplification

We provide an intuition about the tool of *privacy amplification* [BBR88], which is applied during the protocol (i.e. Step 5 of Protocol 3.1) to ensure that any leaked information to a dishonest party is removed, and thus security can be guaranteed. Suppose that Alice holds a random bit-string A of length n about which a dishonest Bob has partial information.[6] This partial information may allow Bob to guess A, since A does not look completely random to Bob anymore. Then privacy amplification is performed by using some randomly chosen two-universal hash function. In short, a hash function is a function that maps a set of some size to a smaller set [CW79]. We say that a class $\mathcal{F}$ of functions $f : \{0,1\}^n \to \{0,1\}^l$. is two-universal if $P_{f \in_R \mathcal{F}}[f(x) = f(x')] \leq s^{-l}$ for all $x \neq x' \in \{0,1\}^n$ [CW79]. Note that $f \in_R \mathcal{F}$ means that f is randomly chosen from $\mathcal{F}$. Then, by applying a randomly chosen two-universal hash-function to her n-bit string A, Alice obtains a shorter string, about which Bob has a negligible amount of information. In particular, the probability that Bob guesses Alice's shorter string is almost uniformly random [Erv+14]. In other words, by shrinking string A in this way, privacy amplification allows for removing the leaked information and thus increases the level of security.

3.3 Protocol for 1-2 OT

Several protocols for 1-2 OT in the noisy-storage model have been suggested and research distinguishes two types of attacks from the adversary in their security proofs: individual and general storage attacks. When the adversary is allowed to only perform measurements on each incoming qubit individually, we call this an *individual* attack. However, since multi-qubit measurements are possible with current technology, it is more realistic to assume that the attacker can do any arbitrary attack, i.e. a *general* attack [KWW12]. Whereas the protocols in [WST08] and [STW09] are proven to be secure only against individual storage attacks, the

[6] More precisely, in our protocol A is the string $x^A{\restriction}_{I_{\bar{c}}}$.

protocols in [KWW12] and [Sch10] allow the dishonest party to do any attack. The latter two protocols are hence much more realistic to consider.

In particular, the protocols in [KWW12] and [Sch10] are proven to be information-theoretically secure against general noisy-storage attacks as long as the laws of quantum mechanics hold and the amount of noise in the quantum storage channel is above a particular level. Recall from Section 1 that an information-theoretically secure protocol implies that there are no restrictions on the attacker's computational power, which is in contrast to many classical cryptographic protocols that are based on computational assumptions and hence more restricted. Although the protocol in [KWW12] requires less storage noise for security, the protocol in [Sch10] is more straightforward and, therefore, simpler to implement. For this reason, we have decided to implement the protocol given in [Sch10].

Protocol for 1-2 OT in the noisy-storage model

Before we provide Protocol 3.2 for 1-2 OT in the noisy-storage model, we provide a protocol for *randomised* 1-2 OT (1-2 ROT), because, as will be further clarified, 1-2 OT can be easily obtained from 1-2 ROT [Sch10]. In 1-2 ROT, Alice has no input whilst Bob holds his choice bit $c \in \{0, 1\}$. After completion of the protocol, Alice holds two bit-strings of the same length, s_0 and s_1, and Bob holds s_c, i.e. the string received by Alice corresponding to his choice bit c. See Figure 2 for an illustration.

We first explain some notation, for which we use some terminology from Section 2. Note that before running Protocol 3.1 and Protocol 3.2, Alice and Bob agree on the length ℓ of the strings s_0 and s_1. The level of security[7] is also decided beforehand, and determines the number n of qubits that will be transmitted in Step 1. Moreover, note that $X \in_R Y$ means that X is *randomly* picked from Y. We also note that H in Step 1 is the Hadamard operator, and thus if $y_i^A = 1$, Alice first applies H to bit x_i^A (for $i \in [n]$) before she sends it to Bob. For clarification, this implies that in Step 1, Alice randomly sends one of the four BB84 states $(|0\rangle, |1\rangle, |+\rangle, |-\rangle)$ to Bob. Finally, in Step 2, when Bob measures his incoming qubit in the Hadamard basis, we mean that Bob first applies H, and then measures in the computational basis.

Protocol 3.1 below is the protocol for 1-2 ROT as provided in [Sch10]. This protocol originates from [Ben+92b; Dam+09], and security of this protocol in the noisy-storage model is proven in [Sch10].

[7] With the level of security we refer to the security error-probability ϵ for an ϵ-secure 1-2 ROT protocol as defined in [Sch10].

Protocol 3.1. [Ben+92b; Dam+09] 1-2 ROT^l

1. Alice randomly picks $x^A \in_R \{0,1\}^n$ and $y^A \in_R \{0,1\}^n$.
   At time $t = 0$, Alice sends $\mathrm{H}^{y_1^A}|x_1^A\rangle, ..., \mathrm{H}^{y_n^A}|x_n^A\rangle$ to Bob.

2. Bob randomly picks $y^B \in_R \{0,1\}^n$.
   Bob measures the i-th qubit he receives in the computational basis if $y_i^B = 0$ and the Hadamard basis if $y_i^B = 1$. He obtains outcome $x^B \in \{0,1\}^n$.

Both parties wait time Δt.

3. Alice sends her basis string y^A to Bob.

4. Bob forms the sets $I_c = \{i \in [n] \mid y_i^A = y_i^B\}$ and $I_{\bar{c}} = \{i \in [n] \mid y_i^A \neq y_i^B\}$, where c is his choice bit and $\bar{c}$ the remaining bit.
   Bob sends I_0 and I_1 to Alice.

5. Alice picks two hash functions $f_0, f_1 \in_R \mathcal{F}$, where $\mathcal{F}$ is a class of two-universal hash functions $f : \{0,1\}^n \to \{0,1\}^l$.
   Alice sends f_0 and f_1 to Bob and outputs $s_0 = f_0(x^A{\restriction}_{I_0})$ and $s_0 = f_1(x^A{\restriction}_{I_1})$.[8]

6. Bob outputs $s_c = f_c(x^B{\restriction}_{I_c})$.

Note that, after Step 2, whenever $y_i^A = y_i^B$ for $i \in [n]$, we also have that $x_i^A = x_i^B$, since Bob measures his i-th incoming qubit in the same basis as in which Alice encoded her i-th bit.

Figure 2: In the setting of 1-2 ROT, Bob inputs a bit $c \in \{0,1\}$. After completion, Alice holds $s_0, s_1 \in \{0,1\}^l$ and Bob holds s_c, which is one of Alice's output strings. Before running the protocol, Alice and Bob agree on the length ℓ of the output strings.

As pointed out in [Sch10], 1-2 OT can be achieved from 1-2 ROT. Recall that in the setting of 1-2 OT, Alice has two input bit-strings of length ℓ, say m_0 and m_1, and Bob—holding his choice bit $c \in \{0,1\}$—wants to obtain m_c after completion of the protocol. Below we provide the resulting protocol for 1-2 OT. An illustration of the required steps is provided in Figure 3. We emphasise that a quantum channel between Alice and Bob is only required for achieving 1-2 ROT, i.e. in Step 1. Moreover, we use the same notation as for Protocol 3.1 and ℓ and n are again determined beforehand.

[8] If the length of $x^A{\restriction}_{I_0}$ or $x^A{\restriction}_{I_1}$ is $m < n$, then Alice adds $n - m$ 0's to the string before applying f_0 or f_1, respectively.

Protocol 3.2. 1-2 OT^l

1. Alice and Bob run Protocol 3.1 for 1-2 ROT.
   After completion, Alice holds two uniformly random bit-strings s_0 and s_1 of length ℓ, and Bob holds s_c corresponding to his choice bit c.

2. Alice computes $c_0 = m_0 \oplus s_0$ and $c_1 = m_1 \oplus s_1$, and sends them to Bob over the classical channel.

3. Bob receives c_0 and c_1. Bob now obtains m_c by computing $s_c \oplus c_c = s_c \oplus (m_c \oplus s_c) = m_c$.

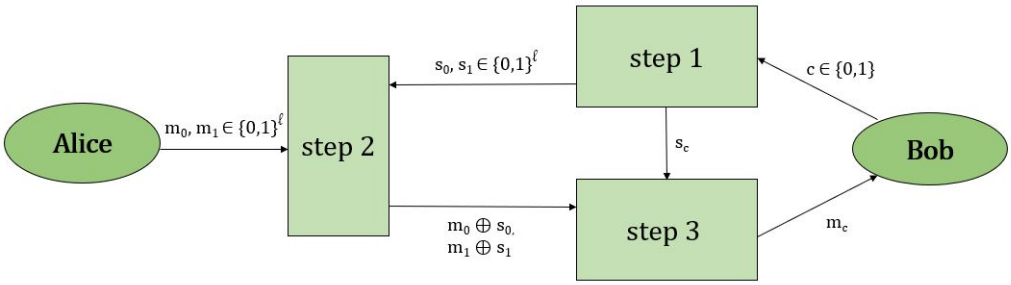


Figure 3: In the setting of 1-2 OT, Alice inputs two strings $m_0, m_1 \in \{0,1\}^l$ and Bob inputs a bit $c \in \{0,1\}$. The three indicated steps are those needed to achieve 1-2 OT from 1-2 ROT. After completion, Bob holds m_c, Alice's input string corresponding to his choice bit c. Contrary to the setting of 1-2 ROT, Alice now does not have an output.

For an explicit analysis of the correctness and a full proof of security of Protocol 3.1 and Protocol 3.2 in the noisy-storage model, we refer the reader to [Sch10].

4 Error correction and robust 1-2 OT

In the practical implementation of 1-2 OT it is realistic to assume that the quantum channel connecting Alice and Bob is subject to a particular level of noise. Specifically, imperfections occur when a quantum channel is implemented in practice, resulting in errors during the transmission of quantum information over the channel [Ben+92a; BS94; EMMM11; GPCZ18]. Apart from the transmission of the qubits itself, the preparation of the qubits by Alice and the measurements by Bob will also be affected by noise [KWW12]. Therefore, we should make the protocol for 1-2 OT *robust* against noise for the honest parties [STW09] so that the protocol is suitable for real implementations. In other words, we need to find a way to deal with the errors resulting from the noisy communication channel and the imperfections during the preparation and measurement of quantum states. This section, in particular, will describe how *information reconciliation*—an additional step during which error-correction techniques are applied—can provide a solution, and subsequently explain the resulting protocol for robust 1-2 OT in the noisy-storage model.[9]

Information theory

Before continuing, we formalise some notions of information theory that will be used in this section. We use the definitions provided in [DS17]. We assume that the reader is familiar with the basic notions of probability theory; however, the relevant definitions are provided in Appendix A.

An important measure of uncertainty about information is the Shannon entropy. We provide three definitions below.

Definition 4.1. (Shannon entropy)
Let X be a random variable on the set $\mathcal{X}$. We define the *(Shannon) entropy* of X as

$$H(X) := -\sum_{x\in\mathcal{X}} P_X(x)\cdot\log_2 P_X(x).$$

In other words, $H(X)$ measures the amount of uncertainty about X: the higher the uncertainty about X, the higher $H(X)$. Note, however, that it is a function of P_X, not of X itself.

Definition 4.2. (Conditional entropy)
Let X and Y be two random variables on the sets $\mathcal{X}$ and $\mathcal{Y}$, respectively. Then, the *conditional entropy* of Y given X is the average uncertainty about Y when the outcome of X is known, i.e.

$$H(Y|X) := -\sum_{y\in\mathcal{Y},x\in\mathcal{X}} P_{YX}(y,x)\cdot\log_2 P_{Y|X}(y|x).$$

Definition 4.3. (Mutual information)
The *mutual information* of two random variables X and Y is defined as

$$I(Y;X) := H(Y) - H(Y|X) = H(X) - H(X|Y).$$

In words, $I(Y;X)$ measures the reduction in uncertainty about Y when X is known, and vice versa.

[9] Note that a noisy communication channel should not be confused with a noisy quantum memory (as in the noisy-storage model), as these notions have a completely different meaning. In particular, if Alice and Bob are both honest, they do not even need a (noisy) quantum memory in order to perform 1-2 OT; however, this does not imply that the quantum *channel* connecting Alice and Bob should be noiseless: errors can still occur and affect the qubits that are sent from Alice to Bob during the protocol.

To provide some intuition, suppose that X is the input of a communication channel and Y the output. Then the mutual information measures how much information output Y reveals about input X. We will now provide a formal definition of a (discrete) channel:

Definition 4.4. (Discrete channel)
A *(discrete) channel* is a triple $(\mathcal{X}, P_{Y|X}, \mathcal{Y})$ consisting of two finite sets $\mathcal{X}$ and $\mathcal{Y}$ and a conditional probability distribution $P_{Y|X} : \mathcal{Y} \to [0, 1]$. Here, $\mathcal{X}$ represents the set of possible channel inputs and $\mathcal{Y}$ the set of possible channel outputs. $P_{Y|X}(y|x)$ is the probability of receiving output $y \in \mathcal{Y}$ when input $x \in \mathcal{X}$ is given.

Definition 4.5. (Memoryless channel)
A channel is called *memoryless* if the probability distribution of the output depends only on the current input.
In other words, if the memoryless channel is used multiple times, previous inputs and outputs do not affect the current output.

Definition 4.6. (Noiseless channel)
A channel $(\mathcal{X}, P_{Y|X}, \mathcal{Y})$ is *noiseless* if both $H(X|Y) = 0$ and $H(Y|X) = 0$.
In other words, the output of the channel completely determines the input, and vice versa.

4.1 Noisy communication channel

Before we introduce the concept of information reconciliation and describe the protocol for robust 1-2 OT, we first discuss in more detail the concept of a noisy communication channel. We consider the case that Alice sends a classical bit-string, A, to Bob over a communication channel. It is often assumed that the communication channel is *reliable* or *noiseless*, i.e. $A = B$, where B is the bit-string received by Bob. However, in real implementations, Alice's string may be affected by noise (i.e. errors) during transmission so that $A \neq B$. In the presence of such noise, we say that the communication channel is *unreliable* or *noisy*. Clearly, if Alice and Bob transmit information over an unreliable communication channel, the correctness and security of the corresponding cryptographic protocol may be affected [GPCZ18].

A noisy communication channel can be modelled by a *binary symmetric channel* (BSC) [BS94]. A BSC is a communication channel in which each transmitted bit is flipped (i.e. affected by noise) with a certain probability p [Mac03]. In other words, in the case that Alice sends her bit-string to Bob, each bit is transmitted correctly with probability $1 - p$. Formally, i.e. according to Definition 4.4, we define a BSC with parameter p by $\mathcal{X} = \mathcal{Y} = \{0, 1\}$ and by

$$P_{Y|X}(0|0) = P_{Y|X}(1|1) = 1 - p,$$

$$P_{Y|X}(0|1) = P_{Y|X}(1|0) = p,$$

where $p \in [0, 1/2]$ [DS17]. We will use BSC(p) to refer to a BSC with probability parameter p. Note that a BSC is memoryless, as the probability that a transmitted bit flips (i.e. p) does not depend on any previous bit that has been transmitted over the channel. Moreover, note that the channel is noiseless if $p = 0$. An illustration of the BSC is provided in Figure 4.

4.2 Information reconciliation

As aforementioned, the presence of noise requires an additional step during the protocol in which Bob can recover the information that Alice has sent so that the correctness of the cryptographic protocol can still be achieved. In the setting of 1-2 OT, this implies that Bob eventually receives Alice's input string m_c corresponding to his choice bit c, even if the quantum channel is noisy.

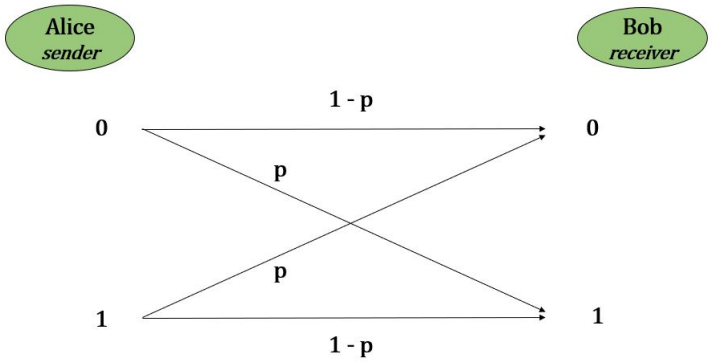


Figure 4: A binary symmetric channel between Alice and Bob, where each bit that Alice sends to Bob is flipped with probability p.

During this additional step, the errors between the transmitted and received information are identified and error correction is performed. In the context of QKD, this process of finding the differences between the strings and performing error correction is often called *information reconciliation* or, simply, *reconciliation* (e.g. in [Ben+92a; BS94; EMMM11]). Information reconciliation occurs via a public classical channel connecting the two parties [Ben+92a; EMMM11].

We provide an simple example of an information-reconciliation protocol by adjusting the description in [BS94] to the setting of 1-2 OT. Suppose again that Alice sends a bit-string A to Bob over a noisy communication channel, and Bob receives the bit-string B. As B is affected by noise, we may assume that $B \equiv A + N \pmod 2$, where the bit-string N represents the noise. In the setting of 1-2 OT, an information-reconciliation protocol aims to generate an estimate, $\hat{A}$, of Alice's string A, given B. As we will see, this string $\hat{A}$ is obtained by exchanging some classical information over a public channel connecting Alice and Bob. At the end of the reconciliation protocol, Bob holds the string $\hat{A}$. Clearly, the protocol should minimise the amount of information revealed to a potential adversary and the probability that the protocol fails to produce the final string (i.e. $\hat{A}$) should be low [BS94].

Privacy amplification

Nevertheless, it is important to note that—by transmitting some extra information over a public channel—information reconciliation allows a dishonest party to gain additional information. Recall the tool of privacy amplification discussed in Section 3.2, which allows us to reduce the amount of information leaked to a potential dishonest party. In particular, since there is additional information leaked during the information-reconciliation protocol, we should perform more privacy amplification to achieve a desired level of security than would be the case if the quantum channel is noiseless and we need not perform information reconciliation. For example, suppose that there are m bits of additional information sent during information reconciliation, then the hash function should map the initial string to a string that is m bits shorter than in the case that the channel is noiseless.

Error-correcting codes

Shannon [Sha48] described that reliable communication over a noisy channel can be obtained by means of encoding and decoding. We refer to an encoding and decoding system as an *error-correcting code*. Intuitively, we may describe encoding and decoding as follows. Instead of immediately sending her message A to Bob, Alice first encodes her message into T (the so-called *codeword*), by adding some *redundancy* to A [Mac03]. Subsequently, T is transmitted over the noisy channel, which outputs the received codeword $R \equiv T + N \pmod 2$. Here, N is a bit-string representing the noise induced by the channel. The decoder then uses the (known) redundancy in order to translates R into the bit-string $\hat{A}$, Bob's estimate of A. Figure 5 provides an illustration.

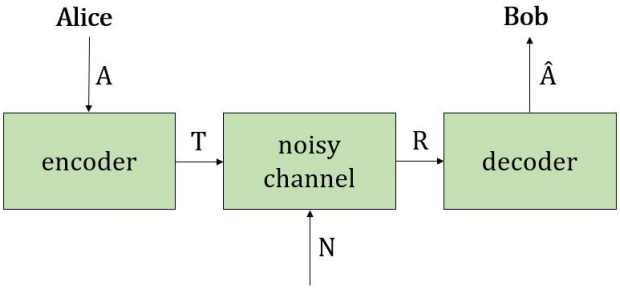


Figure 5: An illustration of an error-correcting communication system for the setting that Alice wants to send a message A to Bob over a noisy communication channel.

Shannon's noisy-channel coding theorem

It is important to mention that there is an upper bound on the amount of reliable communication that can be achieved in the presence of a noisy communication channel. This limit is known as Shannon's noisy-channel coding theorem [Sha48], and hence also referred to as the *Shannon limit*. Before we state the theorem, we provide two more definitions, obtained from [DS17].

Definition 4.7. (Information rate)
The *information rate* R is the number of bits of information that are transmitted per channel use.

Definition 4.8. (Channel capacity)
Let $(\mathcal{X}, P_{Y|X}, \mathcal{Y})$ be a discrete, memoryless channel. Then its *channel capacity* C is defined as

$$C := \max_{P_X} I(X;Y).$$

In other words, the channel capacity measures the maximum amount of mutual information between the input X and the output Y of a the channel.

Theorem 4.1 below shows that information can only be transmitted with arbitrarily small error-probability over a given channel as long as the channel capacity is not exceeded. Theorem 4.1 is a slightly adapted version of the one provided in [DS17], and we refer the reader to this work for the proof.

Theorem 4.1. ***Noisy-channel coding theorem***
Consider a discrete memoryless channel with capacity C. Then, (i) for any rate $R < C$, there is a method of encoding such that information can be transmitted over the channel with negligible maximal error-probability (i.e. the maximal error-probability approaches 0 as the number of channel uses approaches infinity); and (ii) for any rate $R > C$, it is impossible to find a coding system that achieves such a negligible average error-probability.

In other words, whilst the channel capacity C indicates the maximal information rate that *theoretically* can be realised, Theorem 4.1 shows that it should also be possible *in practice* to achieve any rate $R < C$. We therefore aim to find a coding system that approaches the Shannon limit as much as possible.

4.3 Robust protocol for 1-2 OT

In order to make our protocol for 1-2 OT robust against noise for the honest parties, we should add the extra step of information reconciliation. For our implementation of reconciliation in the robust 1-2 OT protocol, we use a class of error-correcting codes called Reed-Solomon (RS) codes. Before we explicitly describe how information reconciliation works in the setting of 1-2 OT and providing the robust protocol, we briefly introduce the class of RS codes.

Reed-Solomon codes

We provide a high-level description of RS codes obtained from [LC83] and refer the reader to this work for a more explicit explanation. RS codes are a class of codes that were introduced by Reed and Solomon in 1960 [RS60]. Each message consists of elements from the Galois (i.e. finite) field $\mathrm{GF}(p^r)$, for p prime. If the original message has length n and the transmitted codeword length m, then there are $m - n$ redundant symbols, and an RS code can correct up to $\frac{m-n}{2}$ errors.

In the following we briefly explain how encoding and decoding of RS codes is established, and use, for clarity, the same notation as in Figure 5. Each message A of length n is associated with a polynomial of the same length. Then, the transmitted codeword obtained after encoding can be considered as a polynomial $T(x)$ of some length $m > n$. The noisy communication channel then adds some noise $N(x)$ (also called the *error pattern* in [LC83]) to $T(x)$. Hence, we can decompose the received message as $R(x) = T(x) + N(x)$. If all coefficients of $N(x)$ are zero, then there are clearly no errors. However, if there are errors, the information from the received message $R(x)$ is recovered by the decoder. In short, the decoding process consists of four steps [LC83]. First, the *syndrome* of $R(x)$ is computed, which, intuitively, is an indicator of the error pattern (in particular, if the syndrome is zero, we assume that there are no errors). Then, from this syndrome we may determine the so-called *error-location polynomial*. In the third step, the error locations are retrieved by computing the roots of the error-location polynomial, and thus the error pattern (i.e. $N(x)$) is obtained. In the last step, we compute $R(x) - N(x)$ to obtain the transmitted codeword $T(x)$.

Information reconciliation for 1-2 ROT

Having provided some intuition into RS codes, we can now give a complete description of the setting of randomised 1-2 OT when the quantum channel is noisy. Although the channel is noisy, we assume that the noise in the quantum channel is independent and identically distributed (i.i.d.), so each qubit that is transmitted over the channel is affected by noise with the same probability.

Recall from Protocol 3.1 for 1-2 ROT that Alice randomly picks two n-bit strings x^A and y^A, and Bob randomly picks an n-bit string y^B. In this section, we consider the case that Alice sends each bit x_i^A encoded as $\mathrm{H}^{y_i^A}|x_i^A\rangle$ to Bob over a *noisy* quantum channel. Bob measures each received qubit either in the computational ($y_i^B = 0$) or Hadamard ($y_i^B = 1$) basis . We now explain what may go wrong in the noisy setting. Recall that if the channel was noiseless, then for each i such that $y_i^A = y_i^B$ (i.e. Alice sends the qubit in the same basis as in which Bob will measure) we have that $x_i^A = x_i^B$ with probability 1. However, if the channel is noisy, then the noise in the channel may affect the qubits that Bob has received. Therefore, when Bob measures each received qubit in basis corresponding to y_i^B, it is possible that his outcome x_i^B is no longer such that $x_i^A = x_i^B$.

Now, recall that at a certain point during the protocol Bob forms the set I_c containing all indices i such that $y_i^A = y_i^B$, where c is his choice bit. [10] Thus, in the presence of noise, it is possible that $x^A{\restriction}_{I_c} \neq x^B{\restriction}_{I_c}$, whereas equality would necessarily hold if the qubits were transmitted over a noiseless channel. Consequently, classical error-correction techniques need to be applied so that Bob obtains an estimate for $x^A{\restriction}_{I_c}$.

In order to understand how information reconciliation works in this setting, we explain how we can represent the occurring noise. For simplicity, let $A = x^A{\restriction}_{I_c}$ and $B = x^B{\restriction}_{I_c}$. Let n be the length of the bit-strings A and B.[11] We assume that B is affected by noise such that $A \neq B$. Note that since the noisy quantum channel is i.i.d., each of the qubits is affected by noise with the same probability p, and thus (without loss of generality) we can assume that each bit of B is affected by noise with probability p. Hence, we may model the situation by sending A over a BSC with error-probability p. Note, however, that the BSC is not the physical channel, but refers to the theoretical channel model. In fact, the BSC represents a noisy *classical* channel. To make this more precise, note that Alice sends her n qubits to Bob over the physical quantum channel. Only after Bob performs his measurements, he obtains classical information. However, if noise has occurred,it is possible that Bob's classical bit-string B differs from the corresponding classical bit-string A encoded by Alice, i.e./ that $x^B{\restriction}_{I_c} \neq x^A{\restriction}_{I_c}$. These differences can hence be represented by the BSC, even though in practice these classical bits were not transmitted over a classical channel.

As aforementioned, we use RS codes to correct the errors induced by the channel. We note that our method of information reconciliation is slightly different from the one described in the protocol from [Sch10]. In particular, [Sch10] considers the case that Alice sends the syndromes (which are of length $n - k$) of her two n-bit strings $x^A{\restriction}_{I_0}$ and $x^A{\restriction}_{I_1}$ to Bob. Instead, to keep the implementations relatively simpler, in our approach, Alice first encodes her two n-bit strings into codewords of length $m > n$, and then sends the last $m - n$ (redundant) bits of both of the codewords to Bob. Bob will attach the $m-n$ bits that belong to $x^A{\restriction}_{I_c}$ to his n-bit string $x^B{\restriction}_{I_c}$, and then use RS decoding techniques to obtain Alice's string $x^A{\restriction}_{I_c}$. Although our method might be slightly less efficient than the method provided in [Sch10], we note that the correctness and security of the protocol from [Sch10] still apply for our approach. Furthermore, we note that as in our approach $m-n$ additional bits are sent over the classical channel, privacy amplification must be adapted accordingly: i.e. the final length of ℓ of the output strings must be shortened by m bits. Nevertheless, we will assume that when Alice and Bob agree on the values for l and n already take into account this reduction in final string length,

The resulting protocol for information reconciliation is given below. All communication is achieved over a noiseless classical channel. We assume that both Alice and Bob are able to perform RS encoding and decoding and have agreed on a parameter m, which is the length of

[10] We will only describe the situation for I_c, as the situation for $I_{\bar{c}}$ is similar (we only need to replace c by $\bar{c}$).

[11] As in Protocol 3.1, for $d \in \{0, 1\}$, if the size m of I_d is smaller than n, we add $n - m$ zeros to $x^A{\restriction}_{I_d}$ to obtain an n-bit string, as this is according to[Sch10].

the encoded list and determines the maximal number of errors that can be corrected. Recall that given the length n of Alice's and Bob's strings, using RS codes we can correct up to $\frac{m-n}{2}$ errors.

Protocol 4.1. Information reconciliation for 1-2 ROT
We assume that Alice holds two n-bit strings $x^A\restriction_{I_0}$ and $x^A\restriction_{I_1}$, and Bob holds an n-bit string $x^B\restriction_{I_c}$, where c is his choice bit.

1. Alice encodes $x^A\restriction_{I_0}$ and $x^A\restriction_{I_1}$ using an RS encoder and sends the last $m-n$ bits of the encoded strings to Bob.
2. Bob attaches the $m-n$ bits corresponding to $x^A\restriction_{I_c}$ to his n-bit string $x^B\restriction_{I_c}$.
3. Bob uses an RS decoder to correct the errors in his resulting m-bit string, if possible. If the number of errors is greater than $\frac{m-n}{2}$, the protocol is aborted.

Robust protocol for 1-2 OT

We will now provide the protocol for robust 1-2 ROT in the noisy-storage model on which our code is based. 4.2 is a slightly adapted from the protocol in [Sch10]. In particular, the protocol in [Sch10] takes into account the possibility that some of the qubits that are transmitted by Alice will eventually not be detected by Bob, which may result from imperfections in Bob's detection apparatus. However, in order to keep the implementation simple and understandable, we assume that all n qubits arrive at Bob. Therefore, a few steps in the protocol from [Sch10] are omitted. We acknowledge, however, that it would be more realistic to assume that not all qubits are detected by Bob. Nevertheless, we emphasise that the correctness and security analysis from [Sch10] still hold in our case.

We use the notation from Section 3.3 and again assume that Alice and Bob agree on l and a particular security error-probability, which determines n. Alice and Bob also agree on the value for m for information reconciliation.

Protocol 4.2. [Sch10] Robust 1-2 ROT^l

1. Alice randomly picks $x^A \in_R \{0,1\}^n$ and $y^A \in_R \{0,1\}^n$.
2. Bob randomly picks $y^B \in_R \{0,1\}^n$.
3. For each $i \in [n]$, Alice sends her bit x_i^A encoded as $\mathrm{H}^{y_i^A}|x_i^A\rangle$ to Bob.
   If $y_i^B = 0$, Bob measures the i-th incoming qubit in the computational basis. Otherwise, he measures the qubit in the Hadamard basis.

Both parties wait time Δt.

4. Alice sends her basis string y^A to Bob.
5. Bob forms the sets $I_c = \{i \in [n] \mid y_i^A = y_i^B\}$ and $I_{\bar{c}} = \{i \in [n] \mid y_i^A \neq y_i^B\}$, where c is his choice bit.
   Bob sends I_0 and I_1 to Alice.
6. Alice performs privacy amplification: she picks two hash functions $f_0, f_1 \in_R \mathcal{F}$, where $\mathcal{F}$ is a class of two-universal hash functions $f : \{0,1\}^n \to \{0,1\}^l$. Alice sends sends f_0, f_1 to Bob.
   Alice performs her part of information reconciliation: she encodes $x^A{\restriction}_{I_0}$ and $x^A{\restriction}_{I_1}$ using an RS encoder and sends the last $m - n$ bits (say r_0, r_1) of the encoded strings to Bob.
   Alice outputs $s_0 = f_0(x^A{\restriction}_{I_0})$ and $s_1 = f_1(x^A{\restriction}_{I_1})$.
7. Bob performs his part of information reconciliation: Bob attaches r_c to his own string and uses an RS decoder to correct the errors on his bit-string $x^B{\restriction}_{I_c}$. He obtains the corrected bit-string x^B_{cor}.
   Bob outputs $\hat{s_c} = f_c(x^B_{cor})$.

We emphasise that robust 1-2 ROT can be converted into robust 1-2 OT in a similar way as has been done for the idealised setting in Protocol 3.2. We therefore omit the resulting protocol.

5 Quantum internet and SimulaQron

A quantum internet is a quantum communication network that allows for the exchange of quantum information over arbitrarily long distances [Cas18; DW18; DLH17]. One of the key aspects of a quantum internet is that it will allow for *quantum entanglement* between any two points in the world [CCB18; Cas18; DLH17; Kim08]. Quantum entanglement is a type of correlation between two quantum systems, which has two important features that can be used for several applications. The first is that two maximally entangled qubits are perfectly correlated: measuring one quantum state will immediately affect the outcome of measuring the other state as well [DLH17; CCB18; WEH18].[12] The second feature of quantum entanglement—complete privacy— is particularly interesting for security-related tasks: no more than two qubits can be maximally entangled, and hence no third party can participate in the entanglement [WEH18].

5.1 Significance and applications of a quantum internet

Instead of substituting the classical internet, it is expected that a quantum internet will be used next to it [WEH18]. Therefore, we explain the significance of having quantum internet next to the classical internet. Moreover, we explain what the benefits may be of constructing a quantum internet rather than building a large-scale quantum computer. We conclude with some potential applications of a quantum internet.

Comparison with the classical internet and the quantum computer

A quantum network may allow us to enlarge the state space used for quantum computation. In particular, with a quantum network we may not only surpass the capabilities of a classical network, but potentially also those of a single quantum processor.

Firstly, each node in a quantum network consists of a quantum processor, which has several advantages over a classical computer. In particular, a quantum computer allows us to exploit the laws of quantum mechanics—e.g. by making use of quantum entanglement—in order to tackle problems that are hard to solve, or unsolvable, with only classical means. Moreover, the state space of a quantum processor (i.e. the number of states on which it operates) scales exponentially with the number of qubits it entails, as a consequence of the superposition principle [Kim08; CCB18]. Recall from Section 2 that a single qubit can exist in a superposition of two states, and therefore n qubits can exist in a superposition of 2^n states. Therefore, whereas a classical processor consisting of n classical bits has a computational space of dimension n, a quantum processor consisting of n qubits has a computational space of dimension 2^n. Consequently, this larger state space of the quantum processors in the network may allow us to solve problems exponentially faster than would classically be possible [Kim08; WEH18].

However, the potential of a quantum internet is not only explained from the fact that its individual components—i.e. the quantum processors—may perform better than if they would be substituted by classical processors, but also from the fact that *interconnecting* the quantum processors with each other increases the overall computational power of the network. This feature is called "quantum connectivity" by [Kim08] and is explained as follows. Suppose that we have k quantum processors, consisting of n qubits each (i.e. each has computational space of dimension 2^n). Then, if these quantum processors are connected via quantum channels, the overall computational space of the network is of dimension 2^{kn}. In contrast, if these k quantum processors (nodes) would be connected via classical channels, then the computational space is only of dimension $k2^n$. In other words, because of its quantum connectivity, a quantum internet

[12] Note that we consider the Copenhagen interpretation of quantum mechanics.

potentially allows for an improved performance than each of its quantum processors is capable of individually.

In particular, some researchers (e.g. [DLH17]) speculate that the realisation of a quantum internet is more feasible than building a large-scale quantum computer. The development of quantum computers places a maximum on the attainable size of individual quantum processors [Kim08], despite the fact that technological progress may increase this maximum over time. In contrast, a quantum internet is potentially able to exceed this maximum by interconnecting these individual processors, as previously explained. Moreover, many of the potential applications of a quantum network require only a quantum processor consisting of a single qubit, and thus not necessarily a large quantum computer. We may therefore consider quantum internet as a possible (temporary) solution for the challenges that prevail in building a large-scale quantum computer.

Potential applications

The possibility of quantum communication over long distances opens many potential applications for a future quantum internet. One of the main expected applications of a quantum internet is quantum cryptography, and thus a quantum internet can be used to provide secure communication [DLH17]. For example, we could implement QKD [DLH17; WEH18] in order to establish a secure secret key between any two points in the network. Besides QKD, a quantum internet will also allow us to realise protocols for two-party cryptographic tasks [DW18; WEH18].[13] Another application of a quantum internet is distributed quantum computation [DLH17; Cac+18]: by linking different small-sized quantum computers via quantum channels [DLH17], we can scale up the computational space of each individual node, as explained before. Therefore, a quantum internet consisting of multiple quantum processors may allow us to solve problems (exponentially) faster than would be possible with only a single quantum processor. Other applications of a quantum internet include, but are not limited to, clock synchronisation [Kóm+14] and improving the achievable resolution of telescopes [GJC12].

5.2 Towards the realisation of a large-scale quantum network

Having discussed the significance and some potential applications of a future quantum internet, we will now provide a high-level description of how a quantum internet could be constructed and analyse the feasibility of this construction. Before continuing, however, we first describe how quantum information is transmitted.

Transmission of quantum information

The direct transmission of information in a classical or quantum communication network is affected by imperfections, i.e. by signal losses or noise. Classical information is often transmitted as electromagnetic waves over optical or free-space fibres [DLH17]. In the presence of noise or signal losses, the affected classical information can be recovered by reamplifying the signal at a so-called classical repeater [DLH17] or simply by resending the information [WEH18]. However, these classical solutions of amplification and repetition cannot be applied when the transmission of *quantum* information is affected by such imperfections. For direct transmission, quantum information is encoded as photons and, similar to classical communication, these photons are transmitted over optical or free-space fibres [DLH17]. However, if a transmitted photon is either

[13] The realisation of these cryptographic protocols obviously depends on the assumptions imposed by the protocol, e.g. that the adversary's quantum storage is noisy for the realisation of the 1-2 OT protocols in this paper.

lost or affected by noise, then the quantum information it carries is destroyed. More precisely, as a consequence of the no-cloning theorem [Die82; WZ82], which states that we cannot copy the quantum state of a qubit, we cannot recover the lost quantum information by sending it again. As a result, the occurrence of losses and dephasing errors during transmission only allow for sending photons over a few hundred kilometres [DLH17], and hence we need to find a different way to transmit quantum information over longer distances in order to establish a large-scale network.

Although quantum error correction may provide a method to overcome the photon losses and errors, and thus may still enable the direct transmission of photons over long distances [DLH17], further research in this direction is required before we can conclude whether this is an acceptable solution. More precisely, this approach is currently still challenging as it requires a relatively low error rate [DLH17]. Nevertheless, an alternative technique to enable the exchange of quantum information between two remote nodes is proposed and is known as *quantum teleportation* [Ben+93]. Quantum teleportation[14] is a method to transmit an unknown quantum state for which the sender and receiver are solely required to share two entangled qubits and to be connected by a classical communication channel. Hence, quantum teleportation does not require a quantum communication channel between the sender and the receiver at the moment of transmission. In particular, quantum teleportation achieves long-distance transmission of quantum information without physically sending the particle that stores it [Cac+18] and without violating the principles of quantum mechanics [NC10]. Since the quantum information is not transmitted directly (i.e. physically), quantum teleportation prevents that the transmission is affected by noise or photon losses.

Although we have explained that quantum teleportation is a crucial technique to establish a large-scale quantum network, teleporting quantum information between two remote nodes forces these two nodes to be entangled with each other. Once entangled, the distance between the two nodes can be arbitrarily long. However, *establishing* quantum entanglement between the nodes cannot be done over an arbitrarily long distance, since the entanglement distribution rate gradually decreases over distance [Cac+18]. In other words, before we can apply quantum teleportation over arbitrarily long distances we still need to find a way to provide entanglement over such a distance. In the next section, we explain that such distributed entanglement can be obtained by placing intermediate nodes, called *quantum repeaters*, which apply a technique called *entanglement swapping*.

Constructing a quantum internet

Having explained how one can transmit quantum information from one node to another, we will now describe how a quantum internet can be constructed.

The three concrete building blocks for a quantum internet are end nodes, photonic communication channels and quantum repeaters, which will be described below [WEH18]. See Figure 6 for a simplified illustration. The end nodes are quantum processors, which may vary from a single qubit to multiple qubits forming a large-scale quantum computer, depending on the task that must be performed.[15] At the end nodes, quantum states are generated, processed and stored [Kim08]. Moreover, these end nodes are connected to the network via a photonic channel—i.e. a quantum channel—to enable the transmission of (quantum) information and distribute entanglement across the network [Kim08]. Similar to the classical internet, this

[14] For a specific description of the working of quantum teleportation, we refer the reader to Chapter 1 of [NC10].

[15] As will also be pointed out later on, Wehner, Elkouss and Hanson [WEH18] explain that not all applications of a quantum internet require a full-blown quantum internet; sometimes a slightly less developed "version" is already sufficient to perform a certain task.

physical connection can be established via fibre-based channels (e.g. optical fibres), free-space channels, or a combination of them, provided that photon loss and decoherence (i.e. loss of quantum information stored in a qubit over time) is minimal [WEH18].

The third component of a quantum internet is a device that increases the distance through which the quantum information is transmitted: a quantum repeater. More precisely, a quantum repeater enables the distribution of entanglement over a long distance. Consequently, we can take advantage of this established entanglement to "teleport" quantum information over this longer distance. As was explained before, quantum teleportation provides a solution to the imperfections that hinder the long-distance transmission of quantum information via photons. When the distance between two end nodes is too large to establish the entanglement required to perform quantum teleportation, quantum repeaters are positioned along the communication channel as intermediate nodes to bridge this distance. In particular, quantum repeaters are placed in such a way that the distance to each end node is small enough to establish entanglement over the channel [WEH18]. After generating entanglement between itself and each of the end nodes, the quantum repeater implements entanglement swapping (see Figure 7): it teleports the entangled qubit that it shares with one of the end nodes to the other end node, such that the two end nodes eventually share an entangled qubit [WEH18]. In this way, entanglement is established over a distance that could be longer than the maximum distance possible via direct transmission [WEH18].

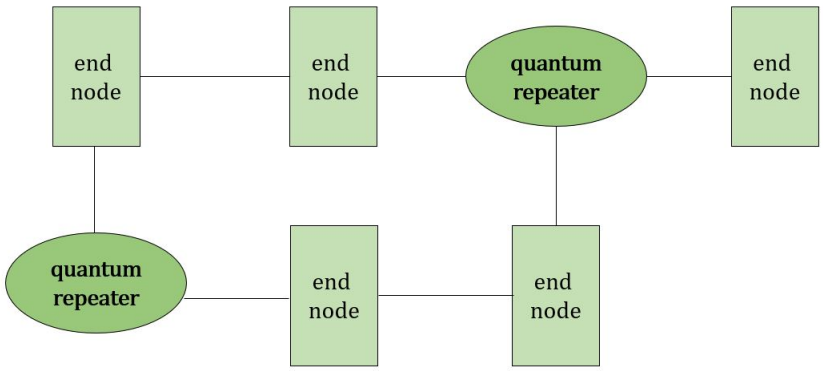


Figure 6: A simplification of the components of a quantum internet. The end nodes are the quantum processors where quantum states are generated, processed and stored. The quantum repeaters function as intermediate nodes, bridging the distances between the end nodes. The end nodes and quantum repeaters are connected via photonic channels (lines), over which quantum information is distributed.

Difficulties of establishing a quantum internet

From the above, it is clear that different issues arise when aiming to establish a quantum internet. These challenges are imposed by quantum mechanical features such as quantum entanglement and no-cloning [Cac+18]. Since these features do not have a classical counterpart, a considerable "paradigm shift" is needed in order to counteract these barriers. An example of such a barrier is the occurrence of errors (e.g. due to decoherence or imperfect operations) when qubits interact with the environment [Cac+18]. Specifically, since quantum information

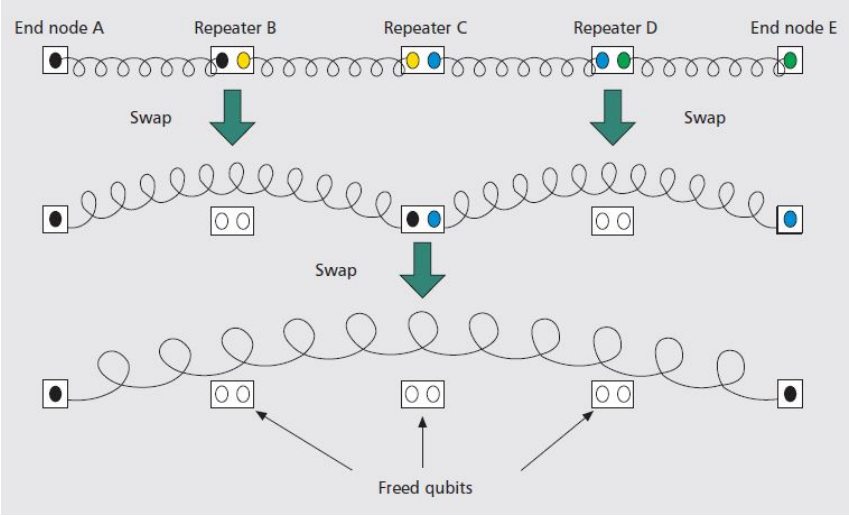


Figure 7: (Source: [MT13]) In order to increase the distance between two end nodes, one or more quantum repeaters are placed along the quantum channel that connects the end nodes. Here, three quantum repeaters are placed between end node A and end node E. For the end nodes and repeaters, entangled pairs are shared between the adjacent nodes. First, repeater B teleports the qubit she shares with A to repeater C using her entanglement with repeater C. After this process of entanglement swapping, node A now shares an entangled pair with repeater C. In a similar way, entanglement swapping between repeater C, repeater D and end node E results in an entangled pair between repeater C and end node E. Then, once more entanglement swapping takes place to ensure that end node A and end node E share an entangled pair. All other qubits are freed.

cannot be copied, we are unable to apply classical error-correction methods that depend on information cloning, and thus a new solution is required [Cac+18]. Nevertheless, research is devoted to overcome the challenges posed by the development of a quantum internet [Kim08] and the realisation seems to be technologically possible, even though we are still at an early stage [DLH17]. Specifically, Wehner, Elkouss and Hanson [WEH18] even predict that a small-scale quantum internet will be established within a few years. Therefore, the current developments seem to provide an optimistic outlook for the establishment of a quantum internet.

Secure two-party computation on a quantum internet

Fortunately, a fully developed world-wide quantum internet is not required to implement protocols for secure two-party computation. In particular, the process of establishing a quantum internet is divided into several developmental stages by [WEH18], including a description of the known applications of a quantum internet that can (already) be established at each stage. This work demonstrates that protocols for secure two-party computation can already to some extent[16] be realised in the stage that allows for end-to-end transmission of qubits as well as the preparation and measurement of a quantum state at any node [WEH18]. Moreover, experimental results show that the essential elements required for this stage—including quantum

[16]Nevertheless, the realisation of these protocols is only possible if the inaccuracies in transmission and measurement and the probability that the prepared quantum state is lost are below a certain level. It is still an open question, however, what the specific bounds are for these three parameters [WEH18].

repeaters—are within reach [WEH18]. Therefore, we may expect that the realisation of secure two-party computation is feasible in a near-future implementation of a quantum internet.

5.3 Simulating quantum internet: SimulaQron

If a quantum internet will be deployed in the future, we will need adequate software for quantum-internet applications. For this reason, the quantum-internet simulator SimulaQron has been created [DW18]. SimulaQron has the purpose to serve as framework in which software for quantum-internet applications can be written and debugged. To the best of our knowledge, SimulaQron is the only framework for developing software for quantum-internet applications currently available.[17]

Working of SimulaQron

We will briefly describe the working of SimulaQron based on Figure 8.[18] The virtual simulation network is established by classically connecting so-called *virtual nodes* with each other. A virtual node is labelled by a name (say Alice) and can be viewed as a server program running on a local classical computer that wants to connect to the network. The network can be simulated locally by running the different server programs all on one physical computer, but a distributed simulation is also possible (i.e. different virtual nodes are connected to different computers). In other words, the virtual nodes corresponding to "Alice Computer", "Bob Computer", and "Charlie Computer" in Figure 8 could either be run on one single computer, but also on two or three distinct classical computers. SimulaQron uses an existing quantum-hardware simulator to simulate the quantum processor on each virtual node, enabling a virtual node to simulate and manipulate qubits. By default, SimulaQron uses the stabilizer formalism[19] as quantum-hardware simulator, but it also supports ProjectQ [SHT18] and QuTip [JNN12].[20]

Besides simulating and manipulating qubits, a virtual node can also connect to other nodes in the network to enable classical and (simulated) quantum communication. This communication between virtual nodes is illustrated in Figure 8 by the lines between the different computers. The SimulaQron servers on the distinct virtual nodes allow for connecting the underlying simulated quantum hardware in order to establish (simulated) quantum communication and quantum entanglement, which is labelled "SimulaQron internal communication" in the figure. The classical-quantum-combiner (CQC) server in each node functions as a link between the SimulaQron back end and the level at which applications are written. In other words, applications on a particular computer (i.e. virtual node) communicate with the SimulaQron server via this CQC server. In addition, as quantum-network applications will often require classical communication as well, SimulaQron also allows for classical communication between distinct virtual nodes, labelled "application communication" in the figure.[21]

[17] Another quantum-network simulator called *NetSquid* is currently under development at QuTech in Delft, but not yet publicly available.

[18] For an extended description of the working of SimulaQron we refer the reader to [DW18].

[19] The stabilizer formalism is a specific approach to simulate so-called stabilizer circuits on a classical computer, which are quantum circuits that are restricted to specific quantum operations. For example, see [Got97] for more information.

[20] The underlying back end can be changed to ProjectQ or QuTip in the settings via `simulaqron set backend`. However, one could also implement another existing simulator as long as it supports working with Python [DW18].

[21] More precisely, this classical communication can be established by opening a socket connection between the nodes. We are allowed to program such a client/server setup in Python, but the Python CQC library of SimulaQron also provides a built-in feature to establish classical communication via the methods `sendClassical` and `recvClassical` (see Section 6.1).

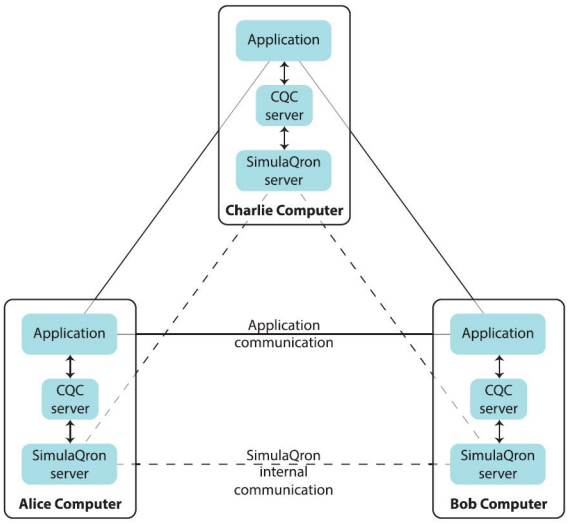


Figure 8: (Source: [DW18]) An illustration of the communication in a quantum network simulated by SimulaQron.

Figure 9 illustrates how we can program quantum-internet applications in SimulaQron. There are two ways to program the quantum network simulated by SimulaQron. The first method—called programming "in native mode"—is by directly assessing the back end of SimulaQron using the Twisted [Twi] library for Python. However, the recommended way is via the provided classical-quantum-combiner (CQC) interface. The CQC interface could be perceived as an intermediate point between the application level and the SimulaQron back end, which allows the user to program at a higher level and hence facilitates the writing of code. Moreover, the CQC interface comes with both a Python and a C library that consists of useful methods to perform quantum operations in Python and C, respectively.

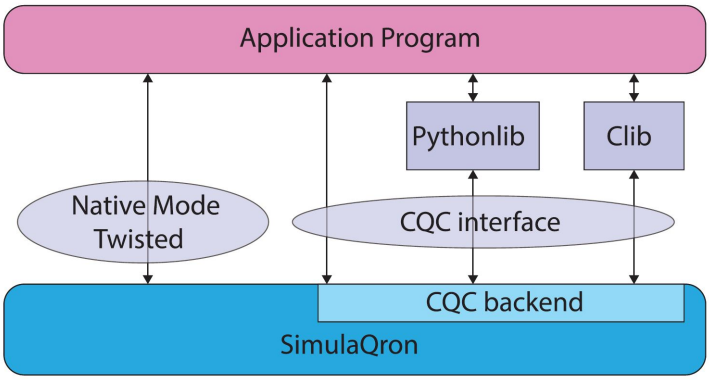


Figure 9: (Source: [DW18]) An illustration of programming a quantum network simulated by SimulaQron, which can be done either by directly assessing the SimulaQron back end using Twisted (left) or by running applications via the CQC interface (right).

Benefits of SimulaQron

Using SimulaQron has several advantages, as explained in [DW18]. First of all, SimulaQron provides software developers with a toolkit to write software that could later be put into practice on a real quantum internet, with little or no adjustments. In particular, application development is independent of the underlying quantum hardware (as a result of the CQC interface) and is promoted by providing programming libraries for Python as well as C. Furthermore, the fact that it is written in Python facilitates a further extension of SimulaQron. Another benefit, which is of particular interest to our work, is that we may implement noise in the channel by turning the setting `noisy-qubits` on. Hence, SimulaQron seems to provide the right tools in order to implement our protocols for 1-2 OT.

Expected limitations

Nevertheless, SimulaQron also creates some drawbacks, which may become a limitation for our implementations. A first drawback of SimulaQron that is relevant to our implementations is that time is not modelled accurately, and hence we cannot simulate time-dependent noise [DW18]. It is therefore important to keep in mind that we cannot use SimulaQron to accurately evaluate the robustness of our protocols. Nonetheless, we are still able to simulate a noisy quantum channel, and hence can still implement our protocols. Another drawback of SimulaQron is that we cannot rely on its security for real usage. More precisely, since the entanglement created with SimulaQron is only a simulation of entanglement, the security guarantees of real entanglement do not hold [DW18]. However, as we do not need to establish entanglement for our protocols, we do not expect this to become a limitation.

6 Implementation of the protocols

6.1 Methods

In this section, we explain how we have implemented the protocols for 1-2 ROT (and 1-2 OT) in the noisy-storage model using SimulaQron and the Python CQC library. The version of SimulaQron we work with is version 3.0.3 and our code is written in Python [Pyt].

Before we continue, we provide our reasoning for the choice of the Python environment for our code. Firstly, SimulaQron is itself written in Python and comes with a Python CQC library that provides us with the right tools to implement our algorithms in the simulated quantum network. Therefore, our code corresponds to the examples provided in the SimulaQron documentation [Sim], making it understandable for current and future SimulaQron users. Secondly, Python is a well-known high-level programming language. Hence, by providing our implementations in Python, we expect that our code is clear to other users, which may allow for further optimisation and use.

How to get started with SimulaQron

Section 5.3 explains how SimulaQron establishes a simulated quantum network. We will now explain in more detail how SimulaQron allows us to program our protocols and which tools of SimulaQron we have used for our implementations.

Before we can run our code we have to configure the simulated quantum network. Since we consider two parties for our protocols, the setup for our implementations only requires two virtual nodes: Alice and Bob. Although SimulaQron allows for a distributed simulation, we use the default configuration of SimulaQron in which the servers run in a centralised setting (i.e. localhost). In order to start the SimulaQron back end that consists of the virtual nodes we run the command `simulaqron start`, which will by default start a network with five nodes (labelled Alice, Bob, Charlie, David and Eve). We use the default back end (i.e. the stabilizer formalism), as this back end is the most efficient in the number of qubits [Sim]. We need to change two of the default settings for our implementations, which should be done before starting the back end. These two settings are `max-qubits`, the number of qubits allowed per quantum register in a virtual node (by default set to 20), and `noisy-qubits`, the option to apply noise to the qubits transmitted over the network (by default turned off).

Furthermore, for our implementations we make use of the functionalities provided by the Python CQC library SimulaQron. As explained in Section 5.3, this library allows us to program our protocols using the CQC interface without needing to access the SimulaQron back end directly. In order to use the library, we need to initialise a `CQCConnection` object, which takes as input the name of the node (say, Alice) to which it corresponds. A `CQCConnection` object enables the node to communicate with SimulaQron and with other virtual nodes. Several methods can be applied to a `CQCConnection` object, of which the following are of use for our work:

1. `sendQubit(q, name)`, which sends qubit `q` to node `name`;
2. `recvQubit()`, which receives a qubit sent to this node;
3. `sendClassical(name, msg)`, which opens a socket connection and sends `msg` (integer or list of integers) to `name`;
4. `recvClassical()`, which receives a classical message sent to this node and closes the socket connection.

The second tool from the Python CQC library that we use is the `qubit` object, which takes as input the corresponding `CQCConnection` (allowing for communication with SimulaQron) and is initialised to be in state $|0\rangle$. The useful methods for a `qubit` object are:

1. `X()`, which applies the X operator to the qubit
2. `H()`, which applies the Hadamard operator to the qubit
3. `measure()`, which measures the qubit and returns the outcome

For example, the following code corresponds to sending qubit `q` in state $|1\rangle$ to node "Bob":

```
# initialise the connection
with CQCConnection("Alice") as Alice:
    # the code
    q = qubit(Alice)
    q.X()
    Alice.sendQubit(q, "Bob")
```

For a more explicit description of the possibilities of SimulaQron and the Python CQC library, we refer the reader to the documentation [Sim].

Construction of the code

Our code is based on Protocol 3.1, 3.2, 4.1, and 4.2. Using the above methods provided by the Python CQC library, we can implement the required quantum and classical information in the protocol, as well as the manipulation of the qubits (i.e. the encoding of Alice's random string x^A as qubits to be sent to Bob). In order to facilitate the computations in the code, Alice's and Bob's strings (`s_0`, `s_1`, `m_0`, and `m_1`) are programmed as Python `list` objects. Furthermore, during the protocol we print several messages to indicate that communication between Alice and Bob has occurred. For the privacy amplification step we use random binary matrices to serve as two-universal hash functions.

Additional methods for robust 1-2 OT

As explained in Section 4, in a realistic setting we may expect that the quantum channel is noisy. In order to implement Protocol 4.2 for robust 1-2 ROT we therefore need to find a way to deal with the errors induced by the channel. As explained, we have decided to work with Reed-Solomon (RS) codes and use the `reedsolo` library [Fil]. We are aware that there are more efficient encoding and decoding techniques available, such as sum-product decoding for low-parity-density-check (LDPC) codes as introduced by [Gal62]. However, there is to the best of our knowledge no suitable open-source implementation of such decoding algorithms that could be easily extended to our setting. Moreover, our implementations are currently not designed with the purpose to be as efficient as possible, and we do not intend to work with very large values for the length of the encoded word. Therefore, we do not expect that using the slightly less efficient Reed-Solomon coding techniques will introduce significant problems.

Furthermore, in order to simulate the noise in the quantum channel, we turn the setting `noisy-qubits` on. However, as also mentioned in [DW18], SimulaQron currently does not allow for an accurate simulation of noise. Although the setting `noisy-qubits` induces some noise in the channel, and the level of noise can be somewhat monitored via the setting `t1`, the resulting noise turned out to be unrealistic: the error-probability[22] approaches $\frac{1}{2}$ as the number of qubits increases, which implies that the received information is completely random.

[22]Note that this error-probability is the probability parameter for the BSC model, as explained in Section 4.1.

Therefore, we have made a change to the underlying SimulaQron code by manually setting the error-probability to be $\frac{1}{10}$, as this would be more plausible in a real setting.

6.2 Results

We have implemented both the idealised and the robust protocols for 1-2 ROT and 1-2 OT in the noisy-storage model using SimulaQron. The code is written in Python and available on GitHub. The source code can also be found in Appendix B. We emphasise that in order to run our code both `simulaqron` and `cqc` need to be installed as specified in the SimulaQron documentation [Sim].

Implementation of the idealised protocols

We have implemented Protocols 3.1 and 3.2 for 1-2 ROT and 1-2 OT, respectively, in the noisy-storage model using SimulaQron. We have tested our code both in the setting of a noiseless quantum channel and in the setting that Alice and Bob are connected by a noisy quantum channel, which is simulated by turning the option `noisy-qubits` on and manually setting the error-probability to be $p = \frac{1}{10}$. In the presence of noise, running our implementation of the idealised protocol results in an incorrect output for Bob. So the idealised protocol breaks down in a noisy setting. We provide a few examples of running our code.[23]

Example 6.1. The first example is our implementation of Protocol 3.1 for 1-2 ROT in the idealised setting. 1-2 ROT is realised by the functions called `Alice_ROT(l, n, waiting_time)` and `Bob_ROT(c, l, n)`. Here, the length of the output lists is `l = 10` and Bob holds choice bit `c = 0`. We also set `n = 100` and the waiting time in seconds between Step 2 and Step 3 is `waiting_time = 2`. Before starting the SimulaQron back, we set the maximal number of qubits to 500, as we want to transmit `n = 100` qubits (i.e. any other number greater or equal to 100 would also work). We therefore run the following commands:

```
$ simulaqron set max-qubits 500
$ simulaqron start
```

We run the code in two separate Python processes,[24] one for Alice and one for Bob.
In Alice's terminal we run:

```
>>> Alice_ROT(l=10, n=100, waiting_time=2)
```

In Bob's terminal we run:

```
>>> Bob_ROT(c=0, l=10, n=100)
```

The resulting prints and outputs are illustrated in Figure 10. As the figure illustrates, Bob has indeed received Alice's output list `s_0`, corresponding to his choice bit `c = 0`.

Example 6.2. In our second example, we show our implementation of Protocol 3.2 for 1-2 OT in the idealised setting.
For 1-2 OT, we wrote two functions called `Alice_OT(m_0, m_1, l, n, waiting_time)` and

[23] Note that our examples have the purpose to show the working of our implementations. In reality the number n of transmitted qubits is very large, much larger than we will consider for our examples.

[24] Our code needs to be run in two separate Python processes in order to enable Alice and Bob to exchange classical information via the `sendClassical` and `recvClassical` commands, because the `sendClassical` method will wait until a socket is set up to a remote node and will wait forever if such a socket is never set up. Nevertheless, starting the SimulaQron back end and changing the settings can be done from within one of the two terminals.

```
>>> from Alice_ROT import Alice_ROT
>>> s_0, s_1 = Alice_ROT(10,100,2)
Alice has sent 100 qubits to Bob.
Both parties wait 2 seconds.
Alice has sent y_A to Bob.
Alice has sent f_0.
Alice has sent f_1.
Alice outputs s_0 and s_1.
>>> s_0
[0, 0, 0, 0, 1, 0, 0, 1, 0, 1]
>>> s_1
[1, 0, 0, 0, 0, 1, 0, 0, 0, 1]
>>>
```

(a) Alice's terminal

```
>>> from Bob_ROT import Bob_ROT
>>> s_c = Bob_ROT(0,10,100)
Bob has sent I_0.
Bob has sent I_1.
Bob outputs s_c.
>>> s_c
[0, 0, 0, 0, 1, 0, 0, 1, 0, 1]
>>>
```

(b) Bob's terminal

Figure 10: 1-2 ROT for Alice and Bob. Alice receives two lists, s_0 and s_1. Bob holds choice bit 0 and receives Alice's output list s_0.

```
>>> from Alice_OT import Alice_OT
>>> m_0 = [0,1,1,0,0,1,0,1,1,0]
>>> m_1 = [0,1,1,1,0,1,1,0,1,1]
>>> Alice_OT(m_0,m_1,10,100,2)
Alice has sent 100 qubits to Bob.
Both parties wait 2 seconds.
Alice has sent y_A to Bob.
Alice has sent f_0.
Alice has sent f_1.
Alice outputs s_0 and s_1.
Alice is finished.
>>>
```

(a) Alice's terminal

```
>>> from Bob_OT import Bob_OT
>>> m_c = Bob_OT(1,10,100)
Bob has sent I_0.
Bob has sent I_1.
Bob outputs s_c.
Bob outputs m_c.
>>> m_c
[0, 1, 1, 1, 0, 1, 1, 0, 1, 1]
>>>
```

(b) Bob's terminal

Figure 11: 1-2 OT for Alice and Bob. Alice inputs m_0 and m_1. Bob holds choice bit 1 and receives m_1.

`Bob_OT(c, l, n)`. Again, we set `l = 10`, `n = 100`, and `waiting_time = 2`. However, Bob now holds choice bit `c = 1` and, since we will now run 1-2 OT, Alice holds two 10-bit lists `m_0 = [0,1,1,0,0,1,0,1,1,0]` and `m_1 = [0,1,1,1,0,1,1,0,1,1]`. We use the same settings as in Example 6.1.
In Alice's terminal we now run:

```
>>> Alice_OT(m_0, m_1, l=10, n=100, waiting_time=2)
```

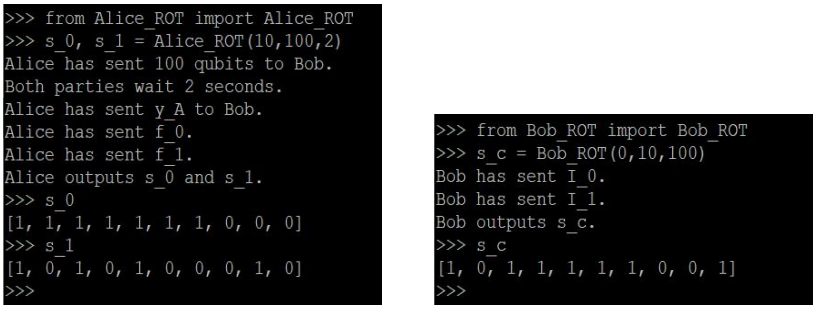
In the second Bob's terminal we run:

```
>>> Bob_OT(c=1, l=10, n=100)
```

The resulting prints and outputs are illustrated in Figure 11. As expected, Bob receives Alice's input list `m_1`, corresponding to his choice bit `c = 1`.

Example 6.3. In our third example, we show what happens when we run 1-2 ROT in a noisy-setting. Hence, we use the functions `Alice_ROT` and `Bob_ROT`. We therefore first have to stop the SimulaQron back end, change the settings, and then start the back end again. We do this by typing the following commands:

```
$ simulaqron stop
$ simulaqron set noisy-qubits on
$ simulaqron start
```



(a) Alice's terminal (b) Bob's terminal

Figure 12: 1-2 ROT for Alice and Bob in a noisy setting. Alice receives two lists, s_0 and s_1. Bob holds choice bit 0, but does not receive the same output as Alice's output list s_0.

Note that this will automatically set the value for `t1` to be 1. We use the same inputs as for our first example. That is, `c = 0`, `l = 10`, `n = 100`, and `waiting_time = 2`.
From Python in Alice's terminal, we run:

```
>>> Alice_ROT(l=10, n=100, waiting_time=2)
```

In the second terminal we run:

```
>>> Bob_ROT(c=0, l=10, n=100)
```

The resulting prints and outputs are illustrated in Figure 12. However, the list received by Bob does no longer correspond to Alice's output list `s_0`. The present noise has affected the output.

Implementation of the robust protocol

In addition, we have considered the case for the robust protocol for 1-2 OT in the presence of noise. We have therefore investigated the noise model that is used in SimulaQron (when turning the option `noisy-qubits` on). As briefly explained before, we may manipulate the error-probability via the setting `t1`, which represents the coherence time of the qubits: the lower the value, the more noise is added to the channel [Sim]. Nevertheless, as we mentioned before, the noise in the channel is unrealistic, regardless of how we tune `t1`, and thus we decided to manually fix the error-probability to be $p = \frac{1}{10}$.

However, regardless of whether we allow noise in the channel or not, the implementation of the robust protocol for 1-2 ROT turned out to be currently infeasible. More precisely, we wrote the two functions `Alice_robust_ROT` and `Bob_robust_ROT` according to Protocol 4.2. Yet, when we run the code, the Python terminals do no longer respond, which occurs more or less halfway the information reconciliation part of the code. In order to find out what may cause the problem, we have experimented with our code. In particular, we have provided a separate code for the part of robust 1-2 ROT during which information reconciliation (i.e. according to Protocol 4.1) is applied. The example below will illustrate that are code for information reconciliation results in the expected outcome and is a working code. However, we note that we obtain the same problem as when running our implementation of Protocol 4.2 if Alice runs her code before Bob (i.e. before we have called the function for Bob). More precisely, the Python terminal again stops responding. Nevertheless, as the code returns the desired outcomes in the case that Bob first runs his code, we expect that the problem is not caused by our use of the `reedsolo` library, but by something else.

From further experimenting with the code, we speculate that the problem is caused by the way classical communication is achieved: via the methods `sendClassical` and `recvClassical`. We recall from Section 6.1 that each time `sendClassical` is called by (say) Alice, a socket connection is opened to Bob, which is only closed after Bob has received his message by `recvClassical`. In fact, note that the information reconciliation protocol requires Alice to send two more classical strings to Bob (i.e. the two $m - n$ strings r_0 and r_1). Therefore, we presume that the problem is caused by an imperfect opening or closing of the socket connection between Alice and Bob during this additional classical information. However, we emphasise the need to further investigate this encountered problem, in order to eventually realise our robust 1-2 ROT implementation.

Despite the fact that running the code currently does not work, we still provide the code of our implementation of the robust protocol for 1-2 ROT in Appendix B. However, as previously explained, we have been able to provide a running code for Protocol 4.1 which establishes the information reconciliation part needed for robust 1-2 ROT. The two implemented functions are called `Alice_reconciliation` and `Bob_reconciliation`. We provide an example of usage below. Recall that for the setting of 1-2 ROT in a *noisy* quantum channel, when information reconciliation is applied Alice has two strings x_0 and x_1, an Bob holds a string $\tilde{x_c}$ for his choise bit c, say 0. In particular, we may assume that the noise in the quantum channel entails that $\tilde{x_c} \neq x_0$. Hence, the task of information reconciliation is to achieve x_c for $c = 0$.

Example 6.4. We illustrate our implementation of Protocol 4.1. Our implementation only requires classical information to be transmitted, and hence we do not need to change any of the default settings and can start SimulaQron immediately:

```
$ simulaqron start
```

Alice holds two binary lists of length 10: namely `x_0 = [0,1,1,1,0,0,1,1,0,1]` and `x_1 = [0,1,0,0,0,1,0,0,1,0]`. Bob holds choice bit `c = 0` and `x_c = [0,1,1,0,0,0,1,0,0,1]`. Note that the lists `x_0` and `x_c` differ in two elements, since we assume that the quantum channel is noisy. Moreover, Alice and Bob agree on `m = 16`. Recall that RS codes can correct up to $\frac{m-n}{2} = 3$ errors, i.e. we may expect that the `reedsolo` codes from [Fil] will be able to correct the two errors on Bob's `x_c`. Indeed, that is what we see below.
In Alice's terminal we run:

```
>>> Alice_reconciliation(x_0,  x_1, m=16, n=10)
```

In Bob's terminal we run:

```
>>> Bob_reconciliation(c=0, x_c, m=16, n=10)
```

The resulting prints and outputs are illustrated in Figure 13. As the figure illustrates, Bob returns Alice's input list `x_0`, corresponding to his choice bit `c = 0`.

```
>>> from Alice_reconciliation import Alice_reconciliation
>>> x_0 = [0,1,1,1,0,0,1,1,0,1]
>>> x_1 = [0,1,0,0,0,1,0,0,1,0]
>>> Alice_reconciliation(x_0, x_1, 16, 10)
Alice is finished.
>>>
```

(a) Alice's terminal

```
>>> from Bob_reconciliation import Bob_reconciliation
>>> x_c = [0,1,1,0,0,0,1,0,0,1]
>>> Bob_reconciliation(0, x_c, 16, 10)
[0, 1, 1, 1, 0, 0, 1, 1, 0, 1]
>>>
```

(b) Bob's terminal

Figure 13: An illustration of our information reconciliation implementation.

6.3 Discussion and implications

In the previous section we have provided the results of our implementations and some examples. We will now, in more detail, discuss what these results entail.

Potential of SimulaQron

First of all, our implementations elucidate the potential of SimulaQron and some current limitations. SimulaQron turns out to be an accessible platform for users to explore the possibilities of a potential future quantum network. SimulaQron is written in Python and provides a Python CQC library which consists of an extensive number of useful commands that facilitate the user to simulate quantum communication and applying quantum operations. In particular, since a version of this CQC interface is intended to be available on the expected quantum network in the Netherlands in 2020 [DW18], software written for SimulaQron could possibly be extended for use on this real quantum network.

In addition, recall that SimulaQron has been designed with the goal to provide a developmental framework for writing and testing software for quantum-internet applications, among which quantum cryptography. Despite some limitations, our results demonstrate the feasibility of implementing 1-2 OT in SimulaQron, and hence the possibility of extending it to implementing any other secure two-party computation protocol. Furthermore, since the quantum part of the considered protocols is similar to that of the BB84 protocol [BB84] for QKD, our implementations also suggest the possibility of a QKD implementation in SimulaQron. Hence, our results indicate the potential of using SimulaQron as environment for implementing quantum-cryptographic protocols. In particular, our results show that SimulaQron could be used for quantum-cryptographic implementations as long as a realistic time-dependent noise model is not desired for testing and the user (i.e. the writer of the code) has sufficient programming experience. However, note that the security of SimulaQron itself is not comparable to the security of a real quantum internet. Therefore, we emphasise that SimulaQron is intended as a framework to *develop* software for quantum-internet applications such as quantum cryptography, and not to actually use this software.

Nevertheless, we encountered two main problems for our implementations which explain some limitations with respect to SimulaQron. Firstly, the methods provided by the Python CQC library (`sendClassical` and `recvClassical`) to enable classical communication have very limited possibilities. These methods merely allow us to transmit (lists of) integers and, for example, no arrays or Python `str` objects. In addition, we speculate that the current way for

establishing a socket connection between any two nodes for classical communication is causing the problem why our implementation of 1-2 ROT did not fully succeed. Although we are aware that SimulaQron also allows for implementing one's own socket connection [DW18], we consider a built-in feature for classical communication essential, since many quantum-cryptographic application will require the use of classical communication [DW18]. For these reasons, we recommend further updating the possibilities of the Python CQC library to allow for more realistic classical communication.

A second limitation was that noise is not accurately modelled by SimulaQron, as is also explained in [DW18]. Although our results show that the noise model allows us to demonstrate that protocols may break down in the case of a noisy quantum channel, the level of noise it achieves is unrealistic, as we explained before. For this reason, it becomes difficult to demonstrate the working of protocols in a noisy setting. Nevertheless, by manually setting the error-probability to a certain (time-independent) level, we were able to get a more realistic level of noise, and we may expect that this approach may be sufficient for future users as well.

Furthermore, we note that SimulaQron is still at a developmental stage: in the period of this work SimulaQron has been updated from version 1.3 to version 3.0.3. The main difficulties we encountered when using SimulaQron in general occurred during the installing of SimulaQron and when updating to the newer versions. In particular, we encountered problems when installing SimulaQron in Windows and hence recommend future users to avoid installing SimulaQron in Windows, if possible. Moreover, one needs to be careful when using SimulaQron, since small typos could already force one to reset and restart SimulaQron. Although these difficulties have provided significant obstacles during the process of writing our code, the updated newer versions of SimulaQron also led to a more consistent and clearer documentation. In addition, the frequent updates of SimulaQron forced us to be very aware of the any new change in SimulaQron, which resulted in a deeper understanding of the working of SimulaQron and its possibilities.

Quantum internet and quantum cryptography

Our implementations provide an illustration of performing the task of 1-2 OT over a potential quantum internet. Secure two-party computation is one of the potential applications of a quantum internet [WEH18], and this thesis shows how such cryptographic applications may be realised in a quantum network. Moreover, we may expect that our implementations might be of use in a future quantum internet: the protocols on which our code is based are designed in a relatively simple way and our code is written in Python, which is a well-known, high-level language. Nevertheless, we speculate whether we would really need a quantum internet to implement secure two-party computation. In particular, secure two-party computation is already shown to be possible with QKD hardware (e.g. see [Erv+14]). Moreover, it is questionable whether we really need a large-scale network in order to implement the considered protocols in practice. In particular, it is reasonable to consider that secure two-party computation is performed at a short distance [Ben+92b].

In particular, we have not only showed how the idealised setting of 1-2 OT (for the noisy-storage model) can be implemented in a future quantum internet, but also considered a more practical setting—that is, when the quantum communication is affected by noise. Although we have not been able to run our code for robust 1-2 ROT in SimulaQron, we have demonstrated the need for such protocols, since the idealised protocols break down in a noisy setting. Moreover, having provided an explicit description of information reconciliation in the context of 1-2 OT and a working code to establish it, we may expect that it will become feasible to eventually implement (a version of) our code for robust 1-2 ROT in SimulaQron.

Other implications of our work

One of the benefits of our work is that our implementations are all publicly available on GitHub. Also the decoding algorithm[25] that we have chosen to use for the implementation of the robust protocol is publicly accessible. This is in contrast to the (to our knowledge) currently existing implementations of 1-2 OT (namely implemented using QKD hardware), for which the used code are licensed and not publicly available. Therefore, our work is more easily accessible, and could hence be used for future research in this direction and further improving the code.

Moreover, as our implementations are simple, clear, and publicly accessible, our work could be used for learning purposes. More precisely, since we have explicitly described the steps in our work and since our code is properly documented, it may be a good source for newcomers in the field in order to explain both the working of SimulaQron (and hence a future quantum internet) and of secure two-party computation in the setting of a noisy-quantum memory. Previous work, such as in [Sch10] and [KWW12], provides a complete description of 1-2 OT in the noisy-storage model, yet some steps are left away. Although these steps are trivial to experts in the field, our work may be a valuable clarification for newcomers and outsiders. In addition, our code could potentially be used to provide some more examples in the SimulaQron documentation, as the documentation currently consists of a limited number of examples.

[25] However, we note that our code is most likely not optimal, since most techniques previously used are not available for public use but are probably "better" in terms of efficiency. Therefore, we recommend to further improve our code.

7 Conclusion

The two primary aims of this study were (i) to ascertain the suitability of the protocols of [Sch10] for quantum-internet applications and (ii) to investigate the extent to which SimulaQron is an adequate environment for implementing and analysing quantum-cryptographic protocols.

To the best of our knowledge we have provided the first implementations of quantum-cryptographic protocols in SimulaQron. As a result, this thesis elucidates the potential of SimulaQron and some of its limitations. In particular, our implementations show that SimulaQron may be a suitable environment for implementing quantum-cryptographic protocols, and indicate some limitations that may be overcome in the future. In this way we contribute to a further development of SimulaQron, but also to the current development of a quantum internet. More precisely, we have illustrated how quantum cryptography may be established on a future quantum internet. The analysis in our work may thereby be significant as quantum cryptography is one of the expected applications of a future quantum internet. On the other hand, we contributed to the field of quantum cryptography by providing a quantum-internet simulation of protocols for 1-2 OT. Additionally, our work shows that the protocols provided in [Sch10] are indeed adequate for quantum-internet applications and feasible to implement.

7.1 Future research

At the same time, our work results in several potential areas for future research.

Firstly, we emphasise that our code is currently not designed to be efficient, and hence could be further optimised. In particular, future research may be devoted to improving the decoding part in our implementations of robust 1-2 OT. In addition, it would be valuable to further investigate why our code for robust 1-2 OT currently does not work, which may become beneficial for the development SimulaQron as well. Moreover, as previously explained, the current built-in client/server setup for classical communication in SimulaQron is quite restrictive and inefficient, and hence we would advise to provide an improved feature.

Furthermore, it may be noteworthy to consider the simulation of possible adversarial attacks. Although the security of the implemented protocols is already proven for the given parameters, such an imitation of possible attacks may further clarify the setting of secure two-party computation. This could prove useful for a better understanding of these protocols, but may also explain to outsiders of the field the significance and relevance of such quantum-cryptographic protocols.

Another significant research direction may be to compare our implementations with implementations of 1-2 OT using QKD hardware. Although our quantum-cryptographic implementations are (to the best of our knowledge) the first ones in the setting of a quantum internet, there are already experimental implementations of 1-2 OT using QKD hardware (e.g. see [Erv+14]). Therefore, it would be an interesting direction of future research to compare the performance and feasibility of the implementations in both settings. However, it should be noted that—as previously argued—SimulaQron does not allow for realistically compare the time performance, and therefore we recommend to use another simulator (potentially NetSquid) to continue in this direction.

Lastly, SimulaQron is to date and to our knowledge the only quantum-internet simulator. However, as another simulation framework—i.e. NetSquid [Net]—is under development, it would be interesting to compare SimulaQron's working (and limitations) with NetSquid. In particular, we have explained that although our results show that SimulaQron provides a suitable developmental framework for quantum-internet applications, it is not ideal. Therefore it may be interesting to investigate whether NetSquid overcomes the limitations of SimulaQron.

Bibliography

[ALA18] Ebrahim Ardeshir-Larijani and Farhad Arbab. "Reo coordination model for simulation of quantum internet software". In: *Software Technologies: Applications and Foundations. STAF 2018*. Ed. by Manuel Mazzara, Iulian Ober, and Gwen Salaün. Vol. 11176. Lecture Notes in Computer Science. Springer, Cham, 2018, pp. 311–319. ISBN: 9783030047702.

[BB84] C. H. Bennett and G. Brassard. "Quantum cryptography: Public key distribution and coin tossing". In: *Proceedings of IEEE International Conference on Computers, Systems, and Signal Processing*. Bangalore, 1984, pp. 175–179.

[BBR88] Charles H. Bennett, Gilles Brassard, and Jean-Marc Robert. "Privacy Amplification by Public Discussion". In: *SIAM Journal on Computing* 17.2 (1988), pp. 210–229. DOI: 10.1137/0217014.

[Ben+92a] Charles H. Bennett et al. "Experimental quantum cryptography". In: *Journal of Cryptology* 5.1 (1992), pp. 3–28. ISSN: 1432-1378. DOI: 10.1007/BF00191318. URL: https://doi.org/10.1007/BF00191318.

[Ben+92b] Charles H. Bennett et al. "Practical quantum oblivious transfer". In: vol. 576. Springer Verlag, 1992, pp. 351–366. ISBN: 9783540551881.

[Ben+93] Charles H. Bennett et al. "Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels". eng. In: *Physical review letters* 70.13 (1993). ISSN: 1079-7114.

[BS94] Gilles Brassard and Louis Salvail. "Secret-key reconciliation by public discussion". In: vol. 765. Springer Verlag, 1994, pp. 410–423. ISBN: 9783540576006.

[BCS12] Harry Buhrman, Matthias Christandl, and Christian Schaffner. "Complete insecurity of quantum protocols for classical two-party computation". In: *Physical Review Letters* 109.16 (2012), p. 160501. DOI: 10.1103/PhysRevLett.109.160501. arXiv: 1201.0849v2.

[Cac+18] Angela Sara Cacciapuoti et al. "Quantum Internet: Networking Challenges in Distributed Quantum Computing". In: (Oct. 19, 2018). arXiv: 1810.08421v2 [quant-ph].

[CCB18] Marcello Caleffi, Angela Sara Cacciapuoti, and Giuseppe Bianchi. "Quantum internet: from communication to distributed computing!" In: (2018). arXiv: 1805.04360v1.

[CW79] J.Lawrence Carter and Mark N. Wegman. "Universal classes of hash functions". In: *Journal of Computer and System Sciences* 18.2 (1979), pp. 143–154. DOI: 10.1016/0022-0000(79)90044-8.

[Cas18] Davide Castelvecchi. "The quantum internet has arrived (and it hasn't)". In: *Nature* 554 (2018), pp. 289–292. DOI: 10.1038/d41586-018-01835-3.

[DW18] Axel Dahlberg and Stephanie Wehner. "SimulaQron—a simulator for developing quantum internet software". In: *Quantum Science and Technology* 4.1 (2018), p. 015001. DOI: 10.1088/2058-9565/aad56e.

[Dam+09] Ivan Damgaard et al. "Improving the Security of Quantum Protocols via Commit-and-Open". In: *Advances in Cryptology - CRYPTO 2009, LNCS 5677, pages 408-427* (Feb. 23, 2009). arXiv: 0902.3918v4 [quant-ph].

[Dam+05] Ivan B. Damgård et al. “Cryptography in the bounded quantum-storage model”. In: *Proceedings of the 46th IEEE Symposium on Foundations of Computer Science — FOCS 2005* (2005), pp. 449–458. arXiv: quant-ph/0508222v2.

[Dam+07] Ivan B. Damgård et al. “A tight high-order entropic quantum uncertainty relation With applications”. In: *Advances in Cryptology — CRYPTO 2007*. Vol. 4622. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2007, pp. 360–378. arXiv: quant-ph/0612014v2.

[Die82] Dennis Dieks. “Communication by EPR devices”. In: *Physics Letters A* 92.6 (1982), pp. 271–272. DOI: 10.1016/0375-9601(82)90084-6.

[DS17] Yfke Dulek and Christian Schaffner. *Information Theory*. Lecture Notes. University of Amsterdam, Master of Logic. 2017.

[DLH17] Wolfgang Dür, Raphael Lamprecht, and Stefan Heusler. “Towards a quantum internet”. In: *European Journal of Physics* 38.4 (2017), p. 043001. DOI: 10.1088/1361-6404/aa6df7.

[DM04] Stefan Dziembowski and Ueli Maurer. “On Generating the Initial Key in the Bounded-Storage Model”. In: *Advances in Cryptology - EUROCRYPT 2004*. Ed. by Christian Cachin and Jan L. Camenisch. Springer Berlin Heidelberg, 2004, pp. 126–137. ISBN: 1978-3-540-24676-3.

[EMMM11] David Elkouss, Jesus Martinez-Mateo, and Vicente Martin. “Information Reconciliation for Quantum Key Distribution”. In: *Quantum Information and Computation,* 11.34 (2011). arXiv: 1007.1616v2 [quant-ph].

[Erv+14] Christopher Erven et al. “An experimental implementation of oblivious transfer in the noisy storage model”. In: *Nature Communications* 5.1 (2014). DOI: 10.1038/ncomms4418.

[EGL82] Shimon Even, Oded Goldreich, and Abraham Lempel. “A randomized protocol for signing contracts”. In: *Advances in Cryptology-CRYPTO ’82*. Plenum, 1982, pp. 205–210.

[Fil] Tomer Filiba. *reedsolo 0.3*. URL: https://pypi.org/project/reedsolo/.

[Gal62] Robert G. Gallager. “Low-density parity-check codes”. In: *IRE Transactions on Information Theory* 8.1 (1962), pp. 21–28. ISSN: 0096-1000.

[GPCZ18] Ran Gelles, Anat Paskin-Cherniavsky, and Vassilis Zikas. *Secure Two-Party Computation over Unreliable Channels*. Cryptology ePrint Archive, Report 2018/506. https://eprint.iacr.org/2018/506. 2018.

[GV88] Oded Goldreich and Ronen Vainish. “How to solve any protocol problem - an efficiency improvement (extended abstract)”. In: *Advances in Cryptology — CRYPTO ’87*. Ed. by Carl Pomerance. Vol. 293. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 1988, pp. 73–86. DOI: 10.1007/3-540-48184-2_6.

[Got97] Daniel Gottesman. “Stabilizer Codes and Quantum Error Correction”. In: (May 28, 1997). arXiv: quant-ph/9705052v1 [quant-ph].

[GJC12] Daniel Gottesman, Thomas Jennewein, and Sarah Croke. “Longer-Baseline Telescopes Using Quantum Repeaters”. In: *Physical Review Letters* 109.7 (2012). DOI: 10.1103/physrevlett.109.070503.

[JNN12] J.R. Johansson, P.D. Nation, and Franco Nori. "QuTiP: An open-source Python framework for the dynamics of open quantum systems". In: *Computer Physics Communications* 183.8 (2012), pp. 1760–1772. DOI: 10.1016/j.cpc.2012.02.021.

[Kil88] Joe Kilian. "Founding cryptography on oblivious transfer". In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. STOC '88. ACM, 1988, pp. 20–31. DOI: 10.1145/62212.62215.

[Kim08] H. J. Kimble. "The quantum internet". In: *Nature* 453 (2008), pp. 1023–1030. DOI: 10.1038/nature07127. arXiv: 0806.4195v1.

[Kóm+14] Peter Kómár et al. "A quantum network of clocks". In: *Nature Physics* 10.8 (2014), pp. 582–587. DOI: 10.1038/nphys3000.

[KWW12] Robert König, Stephanie Wehner, and Jürg Wullschleger. "Unconditional security from noisy quantum storage". In: *IEEE Transactions on Information Theory* 58.3 (2012), pp. 1962–1984. DOI: 10.1109/TIT.2011.2177772. arXiv: 0906.1030v4.

[LC83] Shu Lin and Daniel J. Costello. *Error Control Coding: Fundamentals and Applications*. Prentice Hall, 1983. ISBN: 013283796x.

[Lo97] Hoi-Kwong Lo. "Insecurity of quantum secure computations". In: *Physical Review A* 56.2 (1997), pp. 1154–1162. DOI: 10.1103/PhysRevA.56.1154.

[LC97] Hoi-Kwong Lo and Hoi Fung Chau. "Is quantum bit commitment really possible?" In: *Physical Review Letters* 78.17 (1997), pp. 3410–3413. DOI: 10.1103/PhysRevLett.78.3410.

[Mac03] David J. C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Pr., Sept. 1, 2003. ISBN: 0521642981.

[May97] Dominic Mayers. "Unconditionally secure quantum bit commitment is impossible". In: *Physical Review Letters* 78.17 (1997), pp. 3414–3417. DOI: 10.1103/PhysRevLett.78.3414.

[MT13] Rodney Meter and Joe Touch. "Designing quantum repeater networks". In: *IEEE Communications Magazine* 51.8 (2013), pp. 64–71. DOI: 10.1109/mcom.2013.6576340.

[Net] NetSquid. URL: https://netsquid.org/.

[NC10] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Pr., Dec. 1, 2010. ISBN: 1107002176.

[Pyt] Python. URL: https://www.python.org/.

[QuT] QuTech. *Quantum Internet and Networked Computing*. URL: https://qutech.nl/roadmap/quantum-internet/.

[Rab81] Michael O. Rabin. *How to exchange secrets by oblivious transfer*. Technical Report TR-81. Harvard University, 1981.

[RS60] I. S. Reed and G. Solomon. "Polynomial Codes Over Certain Finite Fields". In: *Journal of the Society for Industrial and Applied Mathematics* 8.2 (1960), pp. 300–304. DOI: 10.1137/0108018.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126. DOI: 10.1145/359340.359342.

[Sca+09] Valerio Scarani et al. "The Security of Practical Quantum Key Distribution". In: *Reviews of Modern Physics* 81.3 (2009), pp. 1301–1350. DOI: 10.1103/RevModPhys.81.1301. arXiv: 0802.4155v3.

[Sch10] Christian Schaffner. "Simple protocols for oblivious transfer and secure identification in the noisy-quantum-storage model". In: *Physical Review A* 82.3 (2010), p. 032308. DOI: 10.1103/PhysRevA.82.032308. arXiv: 1002.1495v2.

[STW09] Christian Schaffner, Barbara M. Terhal, and Stephanie Wehner. "Robust cryptography in the noisy-quantum-storage model". In: *Quantum Information and Computation* 9.11-12 (2009), pp. 963–996. arXiv: 0807.1333v3.

[Sha48] Claude E. Shannon. "A mathematical theory of communication". eng. In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423. ISSN: 0005-8580.

[Sho95] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM J.Sci.Statist.Comput. 26 (1997) 1484* (Aug. 30, 1995). DOI: 10.1137/S0097539795293172. arXiv: quant-ph/9508027v2 [quant-ph].

[Sim] SimulaQron. URL: http://simulaqron.org/.

[SHT18] Damian S. Steiger, Thomas Häner, and Matthias Troyer. "ProjectQ: an open source software framework for quantum computing". In: *Quantum* 2 (2018), p. 49. DOI: 10.22331/q-2018-01-31-49.

[TL17] Marco Tomamichel and Anthony Leverrier. "A largely self-contained and complete security proof for quantum key distribution". In: *Quantum* 1 (2017), p. 14. DOI: 10.22331/q-2017-07-14-14.

[Twi] Twisted. URL: https://twistedmatrix.com/trac/.

[Weh08] Stephanie Wehner. "Cryptography in a Quantum World". In: (2008). arXiv: 0806.3483v1 [quant-ph].

[WEH18] Stephanie Wehner, David Elkouss, and Ronald Hanson. "Quantum internet: a vision for the road ahead". In: *Science* 362.6412 (2018). DOI: 10.1126/science.aam9288.

[WST08] Stephanie Wehner, Christian Schaffner, and Barbara M. Terhal. "Cryptography from noisy storage". In: *Physical Review Letters* 100.22 (2008), p. 220502. DOI: 10.1103/PhysRevLett.100.220502. arXiv: 0711.2895v3.

[Weh+10] Stephanie Wehner et al. "Implementation of two-party protocols in the noisy-storage model". eng. In: *Physical Review A: Atomic, Molecular and Optical Physics* 81 (2010), urn:issn:1050–2947. ISSN: 1050-2947.

[WZ82] William K. Wootters and Wojciech H. Zurek. "A single quantum cannot be cloned". In: *Nature* 299.5886 (1982), pp. 802–803. DOI: 10.1038/299802a0.

[Yao82] Andrew C. Yao. "Protocols for secure computations". In: *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. IEEE, 1982. DOI: 10.1109/sfcs.1982.38.

[Zho+12] Zong-Quan Zhou et al. "Realization of Reliable Solid-State Quantum Memory for Photonic Polarization Qubit". In: *Physical Review Letters* 108.19 (2012). DOI: 10.1103/physrevlett.108.190505.

A Probability theory

We define some basic notions of probability theory, using the definitions from [DS17].

Definition A.1. (Probability measure)
A probability measure $\mathbb{P}$ is a function $\mathbb{P} : \Omega \to \mathbb{R}_{\geq 0}$ such that

$$\sum_{\omega \in \Omega} \mathbb{P}(\omega) = 1.$$

Definition A.2. (Probability space)
A *(discrete) probability space* is a triple $(\Omega, \mathcal{F}, \mathbb{P})$, where Ω is a non-empty sample space, $\mathcal{F}$ is an event space, and $\mathbb{P}$ is a probability measure.

Definition A.3. (Random variable)
A *discrete random variable* X on a discrete probability space $(\Omega, \mathcal{F}, P)$ is a function $X : \Omega \to \mathcal{X}$, where $\mathcal{X}$ is a (discrete) set.

Definition A.4. (Probability distribution)
Let X be a random variable on the set $\mathcal{X}$. The *probability distribution* of X is a function $P_X : \mathcal{X} \to [0, 1]$ defined as

$$P_X(x) := \mathbb{P}[X = x],$$

where $X = x$ denotes the event $\{\omega \in \Omega | X(\omega) = x\}$. Note that P_X is often called the *marginal* probability distribution of X.

Definition A.5. (Joint probability distribution)
Let X and Y be two random variables defined on the same probability space, with ranges $\mathcal{X}$ and $\mathcal{Y}$, respectively. The pair XY is a random variable and has probability distribution $P_{XY} : \mathcal{X} \times \mathcal{Y} \to [0, 1]$ given by

$$P_{XY}(x, y) := \mathbb{P}[X = x, y = \mathcal{Y}].$$

Definition A.6. (Conditional probability distribution)
If $\mathcal{A}$ is an even such that $P[\mathcal{A}] > 0$, then we define the conditional probability distribution of X given $\mathcal{A}$ by

$$P_{X|\mathcal{A}}(x) := \frac{P[X = x, \mathcal{A}]}{P[\mathcal{A}]}.$$

If Y is another random variable and $P_Y(y) > 0$, then we write

$$P_{X|Y}(x|y) := P_{X|Y=y}(x) = \frac{P_{XY}(x, y)}{P_Y(y)}$$

for the conditional distribution of X given $Y = y$.

B Source code for implementations

We provide the source code for our implementations. The code is also available on GitHub, click here.

Source code for the idealised 1-2 ROT protocol

```
from time import sleep
from numpy import matrix, random
from cqc.pythonLib import CQCConnection, qubit

def Alice_ROT(l, n=100, waiting_time=2):
    """
    Perform 1-2 ROT for Alice and return two random lists of length l.

    Input arguments:
    l            -- integer, length of output lists (must be smaller than n)
    n            -- integer, length of initial x_A and y_A (default 100)
    waiting_time -- integer, number of seconds that Alice and Bob wait after
                    step 2 during the protocol (default 2)

    Output:
    s_0          -- list of l bits
    s_1          -- list of l bits
    """
    # Error handling.
    if l > n:
        raise Exception("Input argument l cannot be greater than n.")

    # (Step 1)
    # Alice randomly picks x_A and y_A in {0,1}^n.
    x_A = [random.randint(2) for i in range(n)]
    y_A = [random.randint(2) for i in range(n)]

    with CQCConnection("Alice") as Alice:
        # Alice sends n qubits (BB84 states) to Bob.
        for i in range(n):
            q = qubit(Alice)
            if x_A[i] == 1:
                q.X()
            if y_A[i] == 1:
                q.H()
            Alice.sendQubit(q, "Bob")
        print("Alice has sent {} qubits to Bob.".format(n))

        # Wait.
        print("Both parties wait {} seconds.".format(waiting_time))
        sleep(waiting_time)

        #(Step 3)
        # Alice sends y_A to Bob.
        Alice.sendClassical("Bob", y_A)
        print("Alice has sent y_A to Bob.")

        #(Step 4)
        # Alice receives I_0 and I_1 from Bob.
        data0 = Alice.recvClassical()
        I_0 = list(data0)
        data1 = Alice.recvClassical()
```

```
        I_1 = list(data1)

        # (Step 5)
        # Alice randomly picks two two-universal (lxn) hash functions
        # f_0, f_1 and sends them to Bob.
        f_0 = [[random.randint(2) for i in range(n)] for j in range(l)]
        f_1 = [[random.randint(2) for i in range(n)] for j in range(l)]
        for i in range(l):
            Alice.sendClassical("Bob", f_0[i])
        print("Alice has sent f_0.")
        for i in range(l):
            Alice.sendClassical("Bob", f_1[i])
        print("Alice has sent f_1.")
        # Construct x_0 = x_A|I_0 and x_1 = x_A|I_1.
        x_0 = []
        for i in I_0:
            x_0.append(x_A[i])
        for i in range(len(x_A) - len(I_0)):
            x_0.append(0)
        x_1 = []
        for i in I_1:
            x_1.append(x_A[i])
        for i in range(len(x_A) - len(I_1)):
            x_1.append(0)
        # Translate x_0 and x_1 into a numpy nx1 matrix for computation.
        x_0 = matrix(x_0).transpose()
        x_1 = matrix(x_1).transpose()
        # Alice computes s_0 = f_0(x_0) and s_1 = f_1(x_1).
        s_0 = f_0*x_0 % 2
        s_0 = [s_0[i,0] for i in range(len(s_0))]
        s_1 = f_1*x_1 % 2
        s_1 = [s_1[i,0] for i in range(len(s_1))]

    print("Alice outputs s_0 and s_1.")
    return s_0, s_1
```

Listing 1: 1-2 ROT for Alice

```
from numpy import matrix, random
from cqc.pythonLib import CQCConnection

def Bob_ROT(c, l, n=100):
    """
    Perform 1-2 ROT for Bob and return list s_c of length l without revealing c.

    Input arguments:
    c           -- integer 0 or 1, Bob's choice bit c
    l           -- integer, length of output list (must be smaller than n)
    n           -- integer, length of initial y_B (default 100)

    Output:
    s_c         -- list of l bits corresponding to list s_c returned by Alice
    """
    # Error handling.
    if c != 0 and c != 1:
        raise Exception("Input argument c must be either 0 or 1.")
    if l > n:
        raise Exception("Input argument l cannot be greater than n.")

    # (Step 2)
    # Bob randomly picks y_B in {0,1}^n.
    y_B = [random.randint(2) for i in range(n)]
```

```
25
26      with CQCConnection("Bob") as Bob:
27          # Bob measures the ith received qubit in basis y_B[i]
28          # and obtains outcome x_B, a list of length n.
29          x_B = []
30          for i in range(n):
31              q = Bob.recvQubit()
32              if y_B[i] == 1:
33                  q.H()
34              m = q.measure()
35              x_B.append(m)
36
37          # Wait.
38
39          # (Step 3)
40          # Bob receives y_A from Alice.
41          basisinfo = Bob.recvClassical()
42          y_A = list(basisinfo)
43
44          # (Step 4)
45          # Bob forms the sets I_c and I_cbar.
46          I_c = []
47          I_cbar = []
48          for i in range(n):
49              if y_A[i] == y_B[i]:
50                  I_c.append(i)
51              else:
52                  I_cbar.append(i)
53          # Bob sends I_0 and I_1 to Alice.
54          if c == 0:
55              I_0 = I_c
56              I_1 = I_cbar
57          else:
58              I_0 = I_cbar
59              I_1 = I_c
60          Bob.sendClassical("Alice", I_0)
61          print("Bob has sent I_0.")
62          Bob.sendClassical("Alice", I_1)
63          print("Bob has sent I_1.")
64
65          # (Step 5)
66          # Bob receives f_0 and f_1 from Alice.
67          # Here, f_i is a list of l lists of size n.
68          f_0 = []
69          for i in range(l):
70              data0 = Bob.recvClassical()
71              f_0.append(list(data0))
72          f_1 = []
73          for i in range(l):
74              data1 = Bob.recvClassical()
75              f_1.append(list(data1))
76
77          # (Step 6)
78          # Construct x_c = x_B|I_c.
79          x_c = []
80          for i in I_c:
81              x_c.append(x_B[i])
82          for i in range(len(x_B) - len(I_c)):
83              x_c.append(0)
84          # Translate x_c into a numpy nx1 matrix for computation.
85          x_c = matrix(x_c).transpose()
86          # Bob computes s_c = f_c(x_c).
```

```
        if c == 0:
            f_c = f_0
        else:
            f_c = f_1
        s_c = f_c*x_c % 2
        s_c = [s_c[i,0] for i in range(len(s_c))]

    print("Bob outputs s_c.")
    return s_c
```

Listing 2: 1-2 ROT for Bob

Source code for the idealised 1-2 OT protocol

```
from cqc.pythonLib import CQCConnection
from Alice_ROT import Alice_ROT

def Alice_OT(m0, m1, l, n=100, waiting_time=2):
    """
    Perform 1-2 OT for Alice, without revealing Alice's other input list to Bob.

    Input arguments:
    m0           -- list of length l consisting of 0s and 1s
    m1           -- list of length l consisting of 0s and 1s
    l            -- integer, length of input lists (must be smaller than n)
    n            -- integer, length of n for ROT (default 100)
    waiting_time -- integer, number of seconds that Alice and Bob wait after
                    step 2 during the protocol for 1-2 ROT (default 2)

    There is no output.
    """
    # Error handling.
    if l > n:
        raise Exception("Input argument l cannot be greater than n.")

    # (Step 1)
    # Alice runs 1-2 ROT.
    s0,s1 = Alice_ROT(l, n, waiting_time)

    # (Step 2)
    # Alice sends (m0 XOR s0) and (m1 XOR s1) to Bob.
    xor_0 = []
    xor_1 = []
    for i in range(l):
        xor_0.append((m0[i] + s0[i]) %2)
        xor_1.append((m1[i] + s1[i]) %2)
    with CQCConnection("Alice") as Alice:
        Alice.sendClassical("Bob", xor_0)
        Alice.sendClassical("Bob", xor_1)

    print("Alice is finished.")
```

Listing 3: 1-2 OT for Alice

```
from cqc.pythonLib import CQCConnection
from Bob_ROT import Bob_ROT

def Bob_OT(c, l, n=100):
    """
    Perform 1-2 OT for Bob and return Alice's input list m_c without revealing c
    .

    Input arguments:
    c           -- integer 0 or 1, Bob's choice bit c
    l           -- integer, length of output lists (must be smaller than n)
    n           -- integer, length of n for ROT (default 100)

    Output:
    m_c         -- list of l bits corresponding to Alice's input list m_c
    """
    # Error handling.
    if c != 0 and c != 1:
        raise Exception("Input argument c must be either 0 or 1.")
    if l > n:
        raise Exception("Input argument l cannot be greater than n.")

    # (Step 1)
    # Bob runs 1-2 ROT.
    s_c = Bob_ROT(c, l, n)

    # (Step 3)
    # Bob receives (m0 XOR s0) and (m1 XOR s1) from Alice.
    with CQCConnection("Bob") as Bob:
        data0 = Bob.recvClassical()
        xor_0 = list(data0)
        data1 = Bob.recvClassical()
        xor_1 = list(data1)

    # Bob computes m_c.
    if c == 0:
        xor_c = xor_0
    else:
        xor_c = xor_1
    m_c = []
    for i in range(l):
        m_c.append((s_c[i] + xor_c[i]) %2)

    print("Bob outputs m_c.")
    return m_c
```

Listing 4: 1-2 OT for Bob

Source code for information reconciliation during robust 1-2 ROT

```
from cqc.pythonLib import CQCConnection
import reedsolo

def Alice_reconciliation(x_0, x_1, m, n):
    """
    Perform Alice's part of reconciliation for 1-2 ROT protocol, using RS codes.

    Input arguments:
    x_0          -- list of length n consisting of bits
    x_1          -- list of length n consisting of bits
```

```
    m             -- integer, length of input lists x_0 and x_1
    n             -- integer, length of input lists x_0 and x_1

    A message is printed when Alice is finished. There is no output.

    NOTE: the number of errors that can be corrected is <= (m-n/2)
    """
    # Error handling.
    if len(x_0) != len(x_1):
        raise Exception("Inputs x_0 and x_1 must be of the same length.")
    if n != len(x_0):
        raise Exception("Input n must be the same length as x_0 and x_1")

    # Initialise code.
    rs = reedsolo.RSCodec(m-n)

    enc_0 = rs.encode(x_0)
    enc_1 = rs.encode(x_1)
    red_0 = enc_0[n:]
    red_1 = enc_1[n:]

    # Send red_0 and red_1 to Bob.
    with CQCConnection("Alice") as Alice:
        Alice.sendClassical("Bob",red_0)
        Alice.sendClassical("Bob",red_1)

    print("Alice is finished.")
    return
```

Listing 5: information reconciliation for Alice

```
from cqc.pythonLib import CQCConnection
import reedsolo

def Bob_reconciliation(c, x_c, m, n):
    """
    Perform Bob's part of reconciliation for 1-2 ROT protocol, using RS codes.

    Input arguments:
    c             -- integer, Bob's choice bit
    x_c           -- list of length n consisting of bits
    m             -- integer, length of input lists x_0 and x_1
    n             -- integer, length of input lists x_0 and x_1

    Output argument:
    x_cor         -- list of length n, corrected version of x_c (Bob's estimate
    for Alice's x_c)

    NOTE: the number of errors that can be corrected is <= (m-n/2)
    """
    # Error handling.
    if c != 0 and c!= 1:
        raise Exception("Input c must be either 0 or 1.")
    if n != len(x_c):
        raise Exception("Input n must be the same length as x_c")

    # Initialise code.
    rs = reedsolo.RSCodec(m-n)

    # Receive red_0 and red_1 from Alice.
    with CQCConnection("Bob") as Bob:
        red_0 = Bob.recvClassical()
```

```
        red_1 = Bob.recvClassical()

    if c == 0:
        red_c = red_0
    else:
        red_c = red_1

    # Error correction.
    enc_B = x_c + list(red_c)
    x_cor = rs.decode(enc_B)

    return list(x_cor)
```

Listing 6: information reconciliation for Bob

Source code for the robust 1-2 ROT protocol

```
from time import sleep
from numpy import matrix, random
from cqc.pythonLib import CQCConnection, qubit
import reedsolo

def Alice_robust_ROT(l, n=20, m=30, waiting_time=2):
    """
    Perform robust 1-2 ROT for Alice and return two random strings of length l.

    Input arguments:
    l            -- integer, length of output lists (must be smaller than n)
    n            -- integer, length of x_A and y_A (default 20 < m)
    m            -- integer, length of RS encoded lists (default 30)
    waiting_time -- integer, number of seconds that Alice and Bob wait after
                    step 3 during the protocol (default 2)

    Output:
    s_0          -- list of l bits
    s_1          -- list of l bits

    Encoding is handled via reedsolo.
    NOTE: We can correct up to (m-n)/2 errors with RS encoding/decoding.
    """
    # Error handling.
    if l > n:
        raise Exception("Input argument l cannot be greater than n.")
    if n > m:
        raise Exception("Input argument n cannot be greater than m.")

    # (Step 1)
    # Alice randomlys picks x_A and y_B in {0,1}^n.
    x_A = [random.randint(2) for i in range(n)]
    y_A = [random.randint(2) for i in range(n)]

    with CQCConnection("Alice") as Alice:
        # (Step 3)
        # Alice sends each send x_A[i] encoded in basis y_A[i] to Bob.
        for i in range(n):
            q = qubit(Alice)
            if x_A[i] == 1:
                q.X()
            if y_A[i] == 1:
```

```
            q.H()
        Alice.sendQubit(q, "Bob")
    print("Alice has sent {} (random) qubits to Bob.".format(n))

    # Wait.
    print("Both parties wait {} seconds.".format(waiting_time))
    sleep(waiting_time)

    # (Step 4)
    # Alice sends y_A to Bob.
    Alice.sendClassical("Bob", y_A)
    print("Alice has sent y_A to Bob.")

    # (Step 5)
    # Alice receives I_0 and I_1 from Bob.
    data0 = Alice.recvClassical()
    I_0 = list(data0)
    data1 = Alice.recvClassical()
    I_1 = list(data1)

    # (Step 6)
    # Alice randomly picks two two-universal (lxn) hash functions
    # f_0, f_1 and sends them to Bob.
    f_0 = [[random.randint(2) for i in range(n)] for j in range(l)]
    f_1 = [[random.randint(2) for i in range(n)] for j in range(l)]
    for i in range(l):
        Alice.sendClassical("Bob", f_0[i])
    print("Alice has sent f_0.")
    for i in range(l):
        Alice.sendClassical("Bob", f_1[i])
    print("Alice has sent f_1.")

    # Alice constructs x_0 = x_A|I_0 and x_1 = x_A|I_1.
    x_0 = []
    for i in I_0:
        x_0.append(x_A[i])
    for i in range(len(x_A) - len(I_0)):
        x_0.append(0)
    x_1 = []
    for i in I_1:
        x_1.append(x_A[i])
    for i in range(len(x_A) - len(I_1)):
        x_1.append(0)

    # Information reconciliation part:
    # Alice sends red_0 and red_1 to Bob.
    # Initialise RS code.
    rs = reedsolo.RSCodec(m-n)
    # Alice encodes x_0 and x_1 and forms red_0, red_0 consisting
    # of the last (m-n) "redundancy" bits of the encoded lists.
    enc_0 = rs.encode(x_0)
    enc_1 = rs.encode(x_1)
    red_0 = enc_0[n:] # n is length of x_0 and x_1
    red_1 = enc_1[n:]
    Alice.sendClassical("Bob", red_0)
    print("Alice has sent red_0.")
    Alice.sendClassical("Bob", red_1)

    # Translate x_0 and x_1 into a numpy nx1 matrix for computation.
    x_0 = matrix(x_0).transpose()
    x_1 = matrix(x_1).transpose()
    # Alice computes s_0 = f_0(x_0) and s_1 = f_1(x_1).
```

```
        s_0 = f_0*x_0 % 2
        s_0 = [s_0[i,0] for i in range(len(s_0))]
        s_1 = f_1*x_1 % 2
        s_1 = [s_1[i,0] for i in range(len(s_1))]

    print("Alice outputs s_0 and s_1.")

    return s_0, s_1
```

Listing 7: robust 1-2 ROT for Alice

```
from numpy import matrix, random
from cqc.pythonLib import CQCConnection
import reedsolo

def Bob_robust_ROT(c, l, n=20, m=30):
    """
    Perform robust 1-2 ROT for Bob and return string s_c of length l without
    revealing c.

    Input arguments:
    c               -- integer 0 or 1, Bob's choice bit c
    l               -- integer, length of output lists (must be smaller than n)
    n               -- integer, length of y_B (default 20 < m)
    m               -- integer, length of RS encoded lists (default 30)

    Output:
    s_c             -- list of l bits corresponding to list s_c returned by Alice

    Encoding and decoding is handled via reedsolo.
    NOTE: We can correct up to (m-n)/2 errors with RS encoding/decoding.
    """
    # Error handling.
    if c != 0 and c!= 1:
        raise Exception("Input argument c must be either 0 or 1.")
    if l > n:
        raise Exception("Input argument l cannot be greater than n.")
    if n > m:
        raise Exception("Input argument n cannot be greater than m.")

    # (Step 2)
    # Bob randomly picks y_B in {0,1}^n.
    y_B = [random.randint(2) for i in range(n)]

    with CQCConnection("Bob") as Bob:
        # (Step 3)
        # Bob measures the ith incoming qubit in basis corresponding to y_B[i]
        # and obtains outcome x_B, a list of length n.
        x_B = []
        for i in range(n):
            q = Bob.recvQubit()
            if y_B[i] == 1:
                q.H()
            m = q.measure()
            x_B.append(m)

        #Wait.

        # (Step 4)
        # Bob receives y_A from Alice.
        basisinfo = Bob.recvClassical()
        y_A = list(basisinfo)
```

```
51
52          # (Step 5)
53          # Bob forms the sets I_c and I_cbar.
54          I_c = []
55          I_cbar = []
56          for i in range(n):
57              if y_A[i] == y_B[i]:
58                  I_c.append(i)
59              else:
60                  I_cbar.append(i)
61          # Bob sends I_0 and I_1 to Alice.
62          if c == 0:
63              I_0 = I_c
64              I_1 = I_cbar
65          else:
66              I_0 = I_cbar
67              I_1 = I_c
68          Bob.sendClassical("Alice", I_0)
69          print("Bob has sent I_0.")
70          Bob.sendClassical("Alice", I_1)
71          print("Bob has sent I_1.")
72
73          # (Step 6)
74          # Bob receives f_0, f_1 from Alice.
75          # Here, f_i is a list of l lists of size n.
76          f_0 = []
77          for i in range(l):
78              data0 = Bob.recvClassical()
79              f_0.append(list(data0))
80          f_1 = []
81          for i in range(l):
82              data1 = Bob.recvClassical()
83              f_1.append(list(data1))
84
85          # (Step 7)
86          # Bob constructs x_c = x_B|I_c.
87          x_c = []
88          for i in I_c:
89              x_c.append(x_B[i])
90          for i in range(len(x_B) - len(I_c)):
91              x_c.append(0)
92
93          # Bob receives red_0 and red_1 from Alice.
94          red_0 = Bob.recvClassical()
95          print("Bob received red_0.")
96          red_1 = Bob.recvClassical()
97          print("Bob received red_1.")
98
99          # Initialise f_c and red_c.
100         if c ==0:
101             f_c = f_0
102             red_c = red_0
103         else:
104             f_c = f_1
105             red_c = red_1
106
107         # Information reconciliation part:
108         # Bob corrects errors on x_B|I_c and output estimated s_c_hat.
109         # Initialise RS code.
110         rs = reedsolo.RSCodec(m-n)
111         # Bob corrects errors.
112         enc_B = x_c + list(red_c)
```

```
113         x_cor = rs.decode(enc_B)
114         x_cor = list(x_cor)
115 
116         # Translate x_c into a numpy nx1 matrix.
117         x_cor = matrix(x_cor).transpose()
118         # Bob computes s_c_hat = f_c(x_cor).
119         s_c_hat = f_c*x_cor % 2
120         s_c_hat = [s_c_hat[i,0] for i in range(len(s_c_hat))]
121 
122     print("Bob outputs s_c_hat.")
123 
124     return s_c_hat
```

Listing 8: robust 1-2 ROT for Bob