POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

Quantum Key Distribution post processing

Resource-efficient application of post-processing protocols

Supervisor
prof. Antonio Lioy
dott. Ignazio Pedone

Candidato
Marco Giulio Lorenzo Pappalardo

Anno accademico 2022-2023

Summary

In the following work, after conducting a thorough analysis of the main Error Correction and Privacy Amplification protocols, I selected those that, based on key performance metrics, were more efficient in terms of performance relative to the resources used. Specifically, I selected the Cascade protocol as the Error Correction protocol and Cellular Automata as the Privacy Amplification algorithm.

The Cascade protocol is an iterative Error Correction scheme that allows Error Correction at different levels of the QKD system, thereby improving the overall system efficiency. The Cascade protocol has been shown to be more efficient than single-pass Error Correction protocols and is particularly effective in correcting errors caused by fluctuations in the quantum channel. In conclusion, the preference for Error Correction protocols goes to the Cascade protocol. This preference is attributed to Cascade's efficiency, not only in terms of resource utilization but also in terms of superior performance compared to Winnow, especially in scenarios with realistic Quantum Bit Error Rate (QBER).

Cellular Automata are mathematical models that can be used to extract a secure key by analyzing the statistical properties of raw QKD data. In Privacy Amplification, the goal is to extract a shorter and secure key from the raw key by eliminating any correlations that may have been introduced by the eavesdropper. Cellular Automata can be used to generate a random sequence that can be combined with the raw key to create a shorter and secure key. In established experimental conditions, the CA algorithm consistently demonstrates significantly reduced execution times compared to algorithms based on LFSR and FFT. This observation highlights the superiority of the CA algorithm in terms of key generation speed and overall algorithm execution speed. In summary, the Privacy Amplification scheme based on Cellular Automata emerges as a robust and versatile solution. Its adaptability, combined with its efficiency, makes it a powerful tool in the field of modern cryptography, capable of improving both the speed and security of Privacy Amplification.

Most of the work done involved developing software capable of simulating the QKD post-processing. The proposed solution creates a QKD post-processing module, composed of two enclosed sub-modules, the Error Correction and Privacy Amplification modules. The system is designed to be used both independently and integrated into a QKD simulator in a softwarized network environment.

Contents

Chapter 1

Quantum computing and communication

In 2019 Google claimed to have achieved quantum supremacy ≪*Quantum processors have thus reached the regime of quantum supremacy. We expect that their computational power will continue to grow at a double-exponential rate: the classical cost of simulating a quantum circuit increases exponentially with computational volume, and hardware improvements will probably follow a quantum-processor equivalent of Moore's law doubling this computational volume every few years.*≫ [1]

In recent years, quantum key distribution (QKD) has emerged as a promising technology for secure communication. Unlike classical cryptography, QKD relies on the principles of quantum mechanics to ensure that any eavesdropping attempt is detectable, thereby providing unconditional security. However, the practical implementation of QKD is subject to several limitations, including the presence of noise and errors in the quantum channel.

To overcome these challenges, post-processing protocols have been developed to extract a secure key from the noisy data obtained through QKD. In particular, the cascade protocol for error correction and the use of cellular automata for privacy amplification have shown great potential in improving the security and efficiency of QKD systems.

The cascade protocol is a multi-step error correction scheme that allows the correction of errors at different levels of the QKD system, thereby improving the overall efficiency of the system. The protocol involves dividing the raw key into several blocks and correcting errors in each block independently. In the first step of the cascade protocol, an error correction code is applied to the raw data to detect and correct errors. If errors remain after the first step, they are corrected in subsequent steps using progressively more sophisticated error correction codes. The cascade protocol has been shown to be more efficient than single-step error correction protocols and is particularly effective in correcting errors that occur due to fluctuation in the quantum channel.

Cellular automata, on the other hand, are mathematical models that can be used to extract a secure key by analyzing the statistical properties of the raw QKD data. In privacy amplification, the goal is to distill a shorter and more secure key from the raw key by eliminating any correlations that might have been introduced by the eavesdropper. Cellular automata can be used to generate a random sequence that can be combined with the raw key to create a shorter and more secure key.

In this master thesis, we will explore the cascade protocol and cellular automata-based post-processing protocols for QKD systems in greater detail. We will begin by discussing the theoretical foundations of QKD, including the principles of quantum mechanics and the basic components of a QKD system. Next, we will review some of the existing post-processing protocols for QKD systems.

We will then focus on the cascade protocol and discuss its implementation and performance in real-world scenarios. We will evaluate the effectiveness of the protocol in correcting errors and

compare it to other error correction protocols. We will also discuss the impact of the cascade protocol on the overall efficiency of the QKD system.

In the second part of the thesis, we will focus on the use of cellular automata for privacy amplification. We will discuss the basic principles of cellular automata and how they can be applied to QKD systems. We will explore the performance of cellular automata-based privacy amplification protocols and compare them to other privacy amplification protocols. In the final part of the thesis, we will compare the performance of the cascade protocol and cellular automata-based protocols to other existing post-processing protocols. We will evaluate their effectiveness in enhancing the security and efficiency of QKD systems and discuss their potential for future developments. Overall, this thesis aims to provide a comprehensive analysis of the cascade protocol and cellular automata-based post-processing protocols for QKD systems and their potential to improve the security and efficiency of QKD systems in practical applications.

1.1 Dirac notation

In order to attain a deeper comprehension of quantum computing, it is essential to establish a mathematical groundwork.

From a mathematical perspective a qubit state can be described as a unit vector in two-dimensional Hilbert space [2]

Also classical bits can be represented as a vector in Hilbert space $\begin{bmatrix}1\\0\end{bmatrix}$ in which the index of the vector is the possible value e.g 1 or 0 and the value in that index is the probability. In our example the probability of the bit having 0 value is 100 % while the probability of it having the 1 value is 0%. Clearly classical bits can only have one possible state, so the probability vector for one bit will always be either $\begin{bmatrix}1\\0\end{bmatrix}$ or $\begin{bmatrix}0\\1\end{bmatrix}$. To simplify the way they are handled, a more compact notation has been introduced: the bra-ket notation. This notation was introduced in 1939 by Paul Dirac [3], hence it is known as Dirac notation or bra-ket notation.

When referring to the probability vector of two bits the resulting vector will be the tensor product of the two bits

$$|01\rangle = \begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$$

In Dirac notation the latter is represented as $|00\rangle$ in which each zero is the value of the single bit. More generally the tensor product of two vector is represented as

$$|ab\rangle = \begin{bmatrix}a0\\a1\end{bmatrix} \otimes \begin{bmatrix}b0\\b1\end{bmatrix} = \begin{bmatrix}a0\begin{bmatrix}b0\\b1\end{bmatrix}\\a1\begin{bmatrix}b0\\b1\end{bmatrix}\end{bmatrix} = \begin{bmatrix}a0b0\\a0b1\\a1b0\\a1b1\end{bmatrix}$$

1.2 Qubits

A qubit, short for quantum bit, is the fundamental unit of quantum information in quantum computing. It serves as the quantum counterpart to the classical bit, which is the basic unit of information in classical computing. They can be realized using various physical systems, such as superconducting circuits, trapped ions, or quantum dots. The physical implementation of qubits involves manipulating quantum properties, such as the spin of an electron or the polarization of a photon, to create and control quantum states. These physical systems require careful engineering

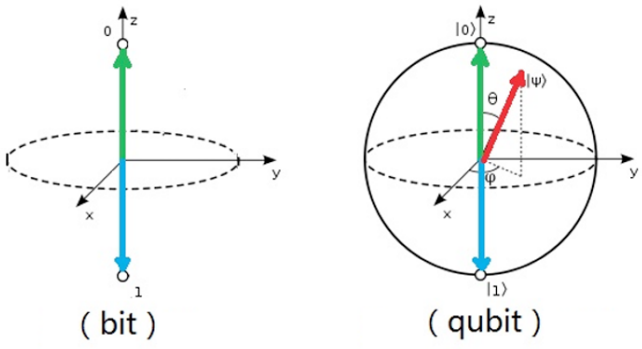


Figure 1.1. Qubit representation on a bloch sphere (source: [4]).

to maintain the fragile quantum coherence and protect the qubit from environmental disturbances that can lead to decoherence and loss of information.

In quantum mechanics, a qubit is typically represented as a linear combination of two ortonormal basis states in Hilbert space, commonly denoted as

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

These basis vectors correspond to the classical states of 0 and 1, respectively. However, unlike classical bits, qubits can be in a state that is a linear combination of $|0\rangle$ and$|1\rangle$.

Mathematically, a qubit's state can be expressed as $\alpha|0\rangle + \beta|1\rangle$, where α and β are complex probability amplitudes. The coefficients α and β determine the probability of measuring the qubit in the state $|0\rangle$ or$|1\rangle$ upon measurement.

It is important to note that the qubit state can be represented using various orthonormal bases, not limited to just $|0\rangle$ and $|1\rangle$. Any pair of orthonormal vectors can be linearly combined to represent the qubit state. However, when measured with the basis $|0\rangle$ and $|1\rangle$, the qubit state collapses to the canonical values of 0 or 1.

Due to the flexibility in choosing the basis, there exists a potentially infinite number of bases to represent qubit states. In addition to the common basis of $|0\rangle$ and $|1\rangle$, other frequently used bases include the Hadamar ones:

$$|+\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \\ \frac{1}{\sqrt{2}} \end{bmatrix} \quad |-\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$$

1.2.1 Superposition

Superposition, as previously described , represents a linear combination of two quantum states. However, beyond its mathematical definition, superposition has a physical interpretation as the addition of two quantum states, resulting in a new quantum state. This concept can be likened to the addition of waves in classical physics, where two waves combine to generate a third wave. Mathematically a qubit its defined in a superposition of states when it's state

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

differs from the orthonormal vector basis

$$|0\rangle = 1|0\rangle + 0|1\rangle$$

and

$$|1\rangle = 0|0\rangle + 1|1\rangle$$

In the context of superposition, each quantum state possesses its own amplitude. When two quantum states are superposed, they interfere with each other in either a constructive or destructive manner. Analogously, in wave terms, constructive interference occurs when two waves with the same phase are added, resulting in the doubling of energy in the resulting wave. On the other hand, destructive interference occurs when two waves completely out of phase are superposed, resulting in the cancellation of energy in the resulting wave.

Quantum interference forms the basis of quantum computing. By leveraging the interference of two particles, it becomes possible to bias the measurement of a qubit towards a desired state. In other words, when a quantum system tackles a problem, it must navigate a pattern of interference where paths leading to incorrect solutions destructively interfere and cancel out, while paths leading to the correct solution constructively interfere, maximizing the energy of the corresponding state representing the correct solution.

It is essential to differentiate quantum interference from decoherence. While interference is an induced phenomenon that produces the desired effect on the qubit state, decoherence pertains to the collapse of the quantum state due to interference with the external environment. Decoherence represents an undesired disturbance on the quantum state, resulting in the loss of superposition properties.

1.2.2 Entanglement

Entanglement is a remarkable phenomenon in quantum computing that arises when multiple qubits become intrinsically correlated, regardless of the physical distance between them. It is a fundamental property that sets quantum systems apart from classical ones. When qubits are entangled, the state of one qubit becomes entwined with the states of the other qubits in the system, resulting in a collective, inseparable quantum state. Mathematically, the entangled state of two qubits can be represented using the tensor product ($\otimes$) notation:

$$|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$$

where $|00\rangle, |01\rangle, |10\rangle$,and $|11\rangle$ are tensor product states representing the basis states of the two qubits. The coefficients α, β, γ, and δ are complex probability amplitudes that satisfy the normalization condition $|\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$. These coefficients determine the probabilities of measuring the qubits in the corresponding basis states upon measurement. Entanglement is not limited to just two qubits, but can occur among any number of qubits. For example, for three qubits, the entangled state can be expressed as:

$$|\psi\rangle = \alpha|000\rangle + \beta|001\rangle + \gamma|010\rangle + \delta|011\rangle + \epsilon|100\rangle + \zeta|101\rangle + \eta|110\rangle + \theta|111\rangle$$

where $|000\rangle, |001\rangle, |010\rangle, \ldots, |111\rangle$ represent the tensor product states of the three qubits. The entangled state cannot be factorized into a simple combination of individual qubit states, indicating the presence of intricate correlations. Entanglement enables highly correlated and interconnected information processing, offering the potential for quantum computers to perform parallel computations and potentially achieve exponential speedup in certain algorithms.

1.3 Quantum gates

Quantum gates can be expressed as matrices and the operation on the qubit is the vector product. Similarly to classical computing, quantum gates are used in quantum circuit to operate on qubits.

Each of them can be represented as a matrix in Hilbert space. We will further discuss the main ones in the following sections

1.3.1 Pauli Gates

The Pauli gates are significant operators that hold a prominent position in our examination of quantum systems. They act on a two-dimensional Hilbert space and can thus be expressed as 2×2 matrices. These matrices play a crucial role in various quantum computations and analyses.

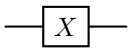
X-Gate

This gate is equivalent to the classical NOT gate and flips the state of a qubit from $|0\rangle$ to $|1\rangle$ or vice versa;

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad X|1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

In quantum circuits the X-Gate is represented as



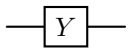
Y-Gate

This gate is similar to the X-Gate but introduces a phase change as well, rotating the qubit state around the Y-axis in the Bloch sphere;

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

$$Y|0\rangle = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ i \end{bmatrix} \quad X|1\rangle = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -i \\ 0 \end{bmatrix}$$

In quantum circuits the Y-Gate is represented as



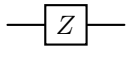
Z-Gate

This gate applies a phase flip to the qubit state, changing the sign of the $|1\rangle$ state while leaving the $|0\rangle$ state unchanged;

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$Z|0\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad Z|1\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

In quantum circuits the Z-Gate is represented as



1.3.2 Hadamar Gate

The Hadamard gate creates superposition by transforming the $|0\rangle$ state to an equal superposition of $|0\rangle$ and $|1\rangle$, and the $|1\rangle$ state to an equal superposition of $|0\rangle$ and $-|1\rangle$ allowing the exploration of multiple computational paths simultaneously. It is a fundamental quantum gate that plays a

crucial role in quantum computing and information processing. Mathematically, the Hadamard gate can be represented as a 2×2 matrix:

$$H = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix}$$

Geometrically, the Hadamard gate corresponds to a rotation on the Bloch sphere by 180 degrees around the axis defined by the linear combination of the $|0\rangle$ and $|1\rangle$ states. This transformation leads to an equal probability distribution of the qubit being measured in either the $|0\rangle$ or $|1\rangle$ basis state.

$$H|0\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \quad H|1\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} \end{bmatrix}$$

In quantum circuits the Hadamar gate is represented as

1.3.3 CNOT Gate

This two-qubit gate performs a NOT operation on the target qubit (flips the state) only if the control qubit is in the $|1\rangle$ state; otherwise, it leaves the target qubit unchanged;

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

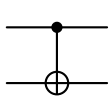
In the following example the control bit (i.e. hte first qubit) qubit is set to $|1\rangle$, meaning that the target qubit will be flipped by the CNOT gate

$$C|10\rangle = C\left(\begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |11\rangle$$

We can also observe that when the control qubit is set to $|0\rangle$, the state of both the qubits is preserved

$$C|00\rangle = C\left(\begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |00\rangle$$

In quantum circuits the CNOT gate is represented as



1.3.4 SWAP Gate

The SWAP gate exchanges the quantum states of two qubits, allowing for the exchange of information or entanglement between qubits;

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$SWAP|10\rangle = C\left(\begin{bmatrix}0\\1\end{bmatrix}\otimes\begin{bmatrix}1\\0\end{bmatrix}\right) = \begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}0\\0\\1\\0\end{bmatrix} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix} = |01\rangle$$

In quantum circuits the SWAP gate is represented as

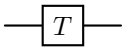
1.3.5 T-Gate

The T-Gate applies a $\pi/4$ phase shift to the $|1\rangle$ state, enabling more complex quantum operations.

$$T = \begin{bmatrix}1&0\\0&e^{\frac{i\pi}{4}}\end{bmatrix}$$

$$T|0\rangle = \begin{bmatrix}1&0\\0&e^{\frac{i\pi}{4}}\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix} \quad T|1\rangle = \begin{bmatrix}1&0\\0&e^{\frac{i\pi}{4}}\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}0\\e^{\frac{i\pi}{4}}\end{bmatrix}$$

In quantum circuits the T-Gate is represented as



1.4 Measurements

In the realm of quantum mechanics, measurement involves manipulating a quantum system to extract a numerical value corresponding to one of its properties. However, this process comes at the cost of forfeiting the original state of the qubit, as it collapses into the measured state.

Practically, measurements are guided by the principles of Born's rule [5], which enables the calculation of the probability that a particular quantum state, represented by $|\psi\rangle$, collapses into a specific state denoted as $|x\rangle$:

$$P(x) = |\langle x|\psi\rangle|^2$$

According to Born's rule, the probability of measuring a quantum system in a particular state is given by the squared magnitude of the corresponding coefficient in the state's superposition.

Consider a qubit with the state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex amplitudes. The probability $P(0)$ of measuring the qubit in the state $|0\rangle$ is given by:

$$P(0) = |\alpha|^2$$

Similarly, the probability $P(1)$ of measuring the qubit in the state $|1\rangle$ is given by:

$$P(1) = |\beta|^2$$

Since $|\alpha|^2$ and $|\beta|^2$ represent the squared magnitudes of the amplitudes, they provide the probabilities of measuring the qubit in the respective states.

To determine the specific values of $P(0)$ and $P(1)$, we need to know the values of α and β from the given qubit state. If we assume that the qubit is in an equal superposition state, where $\alpha = \frac{1}{\sqrt{2}}$ and $\beta = \frac{1}{\sqrt{2}}$, such as after the operation of an Hadamar gate, we have:

$$P(0) = \left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}$$

$$P(1) = \left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}$$

Thus, when the qubit is in an equal superposition state, the probabilities of measuring it in the states $|0\rangle$ and $|1\rangle$ are both 50%.

It is worth to note that the state of the qubit can be different from the one described, and α and β can assume any value as long as $|\alpha|^2 + |\beta|^2 = 1$. This principle gives a hint on the potential of quantum computing, that allows to perform parallel computation at the same time and then set the qubits in a state where the probability of collapsing to a base is higher than the other. It is also important to mention that a when a qubit is measured on a determined basis, any subsequent measurement with that same basis will result in the same outcome independently by the chosen basis.

A different consideration must be made when measuring the same qubit multiple times using different bases. As mentioned, each measurement collapses the qubit state (unless the same basis is used in subsequent measurements). Therefore, if a qubit is measured in one basis and yields a specific outcome with a 50% probability, a new measurement in an orthogonal basis will result in one of the two basis states with a 50% probability once again.

Moreover, the qubit's memory is limited to the most recent measurements performed. For instance, if a qubit is found to be in the $|0\rangle$ state after a measurement, any subsequent measurement in a different basis will nullify that information. Consequently, when the qubit is measured again in the $|0\rangle$ and $|1\rangle$ basis, the measurement outcome will once again be uncertain, with a 50% probability of being either $|0\rangle$ or $|1\rangle$.

Measurement of qubits is done at the end of every computation and to be done, the measurement gate has to be inserted in quantum circuits and it's represented by the following symbol:

1.5 Quantum circuits

Now that we have seen the main components of quantum circuits, we can further discuss how to build and use them. To build a quantum circuit we will be following these steps

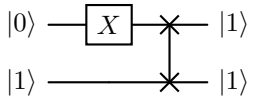
Define the number of qubits Determine the number of qubits required for the computation or experiment.

Initialize the qubits Start with the qubits in a well-defined initial state, typically the $|0\rangle$ state. This is done by applying the X gate or any other gate that transforms the initial state to the desired state.

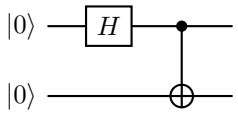
Apply quantum gates Apply a series of quantum gates to the qubits to perform specific operations. These gates can include Pauli gates (X, Y, Z), Hadamard gate (H), CNOT gate, and other single-qubit or multi-qubit gates. These gates manipulate the quantum state and enable quantum computation.

Perform measurements At the end of the circuit, perform measurements on the qubits to extract information. Measurements collapse the quantum state into classical bits, providing the final outcome of the computation.

Like in classical computing, the algorithm we are designing is based on the quantum gates that we apply to the qubits. The following is an example of a circuit that swap two qubits after flipping one



The following example entangles two bits



This circuit can be also represented as

$$CH\left(\begin{bmatrix}1\\0\end{bmatrix}\otimes\begin{bmatrix}1\\0\end{bmatrix}\right)=C\left(\begin{bmatrix}\frac{1}{\sqrt{2}}\\\frac{1}{\sqrt{2}}\end{bmatrix}\otimes\begin{bmatrix}1\\0\end{bmatrix}\right)=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}\begin{bmatrix}\frac{1}{\sqrt{2}}\\0\\\frac{1}{\sqrt{2}}\\0\end{bmatrix}=\begin{bmatrix}\frac{1}{\sqrt{2}}\\0\\0\\\frac{1}{\sqrt{2}}\end{bmatrix}$$

Chapter 2

Quantum Key Distribution

2.1 Quantum Key Distribution

The aim of quantum key distribution is for two individuals, denoted as "Alice" and "Bob," who possess no initial shared confidential data, to arrive at a consensus on a random key that remains undisclosed from an adversary named "Eve," who attempts to intercept their communication. In typical cryptography and information theory, it is assumed that digital communications are susceptible to passive surveillance, allowing an eavesdropper to comprehend the entire message, concealed to the sender and receiver. However, when digital data is encoded in basic quantum systems like single photons, it becomes feasible to establish a communication channel in which the transmissions cannot be reliably decoded or duplicated by an eavesdropper lacking specific information utilized in creating the transmission. The eavesdropper cannot even obtain partial information about such a transmission without causing random and unmanageable disturbances likely to be noticed by the channel's legitimate users.

The fundamental quantum aspect involved, a demonstration of Heisenberg's uncertainty principle, is the presence of pairs of properties that are mutually exclusive, meaning that measuring one property inherently randomizes the value of the other. For instance, measuring a single photon's linear polarization will randomize its circular polarization, and vice versa. In a broader sense, any pair of polarization states will be termed a basis if they correspond to a consistently measurable property of a single photon. Two bases will be termed conjugate if quantum mechanics dictates that measuring one property entirely randomizes the other. BB84 quantum key distribution protocol utilizes two conjugate bases, which we denote as the rectilinear basis (horizontal versus vertical polarization) and the circular basis (left-circular versus right-circular). We shall refer to these as the standard bases. Likewise, a standard polarization can be either horizontal, vertical, left-circular, or right-circular. A third basis, comprising of 45° and 135° diagonal polarizations, also exists, and it is conjugate to both the other two bases. However, we won't need to consider it, except in the context of potential eavesdropping strategies.

The BB84 protocol delineate later remains secure even against an adversary possessing immense computational capabilities (even if $P = NP$!), facing any attack where she is constrained to measure photons (or in the subsequent extension, light pulses) one at a time, and amalgamate the classical outcomes of these measurements with information later overheard during the public discussion (described below). The formalism of quantum mechanics allows a more extensive type of measurement, presently infeasible and foreseeable in the near future. Such a measurement would treat the entire sequence of n photons dispatched during a key-distribution session as a single 2^n-state quantum system, engender it to interact coherently with an intermediate quantum system of similar complexity, sustain the phase coherence of the intermediate system for an arbitrarily extended period, and then eventually measure the intermediate system in a manner contingent on the information overheard during the public discussion. It remains unknown whether the protocol is secure against such an attack, but recent research suggests that it might be [6].

2.2 BB84

The fundamental quantum key distribution protocol (see Table 2.1) commences with Alice transmitting a random sequence of the four standard types of polarized photons to Bob. Bob then independently and randomly selects, for each photon (without knowledge of Alice's choices at this point), whether to measure the photon's rectilinear or circular polarization. Bob subsequently publicly announces the type of measurement he conducted (excluding the measurement outcome), and Alice, again publicly, informs him whether he made the accurate measurement (i.e., rectilinear or circular). Alice and Bob then publicly agree to discard all bit positions for which Bob conducted the incorrect measurement. Correspondingly, they agree to discard bit positions where Bob's detectors failed to detect the photon altogether, a relatively common occurrence with current detectors at optical wavelengths. The polarizations of the remaining photons are interpreted as bit 0 for horizontal or left-circular and bit 1 for vertical and right-circular. The resultant binary string should be confidentially shared information between Alice and Bob, provided that no eavesdropping on the quantum channel has occurred. The outcome of the aforementioned steps is denoted as the quantum transmission (or sometimes the raw quantum transmission to emphasize that it was obtained early in the process).

Alice's random key	0	1	1	0	1	0	0	1
Alice's random basis	+	X	+	X	X	+	X	+
Alice's polarization	↑	↘	→	↗	↘	↑	↗	→
Bob's random basis	+	X	X	X	+	X	+	+
Bob's measurements	↑	↘	↘	↗	→	↗	↑	→
Shared key	0	1	-	0	-	-	-	1

Table 2.1. BB84 protocol.

In the fundamental protocol, Alice and Bob proceed to examine for eavesdropping by openly comparing the polarizations of a randomly selected subset of the photons on which they should agree. Any measurement that the eavesdropper can perform on a photon while it travels from Alice to Bob cannot provide more than $\frac{1}{2}$ expected bits of information about its polarization. Furthermore, any measurement yielding $s \leq \frac{1}{2}$ expected bits has a probability of at least $\frac{s}{2}$ of causing a discrepancy when Bob's and Alice's data are compared, assuming that Bob detects this photon in the correct basis (otherwise, this photon is lost to all parties). If Alice and Bob find no discrepancies, and if it is reasonable to assume that Eve cannot tamper with the content of the public messages exchanged between them, then Alice and Bob can reasonably conclude that there are few or no errors in the remaining non-compared data, and that little or none of it is known to any eavesdropper.

The assumption that the public messages cannot be tampered with by Eve is crucial because otherwise, Eve could position herself between Alice and Bob and impersonate each of them to the other. Consequently, Eve would end up with a string shared with Alice and another one shared with Bob, while Alice and Bob would remain unaware. This critical attribute of the public channel can be implemented in practice either by utilizing an inherently unjammable public channel or by employing an information-theoretically secure authentication scheme [7] to verify that the public messages have not been altered in transit. In the latter case, Alice and Bob need to possess a modest amount of shared secret information in advance to act as an authentication key, and a few bits of this key are made unusable for each instance the key distribution protocol is carried out. However, each successful instance of the protocol provides Alice and Bob with a substantially larger volume of fresh key information, some of which can be utilized to replace the lost authentication bits. Therefore, in this case, the protocol implements key expansion rather than key distribution. It's important to note that in the case of a jammable public channel,

a determined opponent, through repeated interference with either the quantum or public transmissions, could force Alice and Bob to deplete their entire supply of authentication key.

The basic "quality-control" in the basic quantum key distribution protocol, which follows the quantum transmission as described above, is insufficient in practice for two reasons:

1. Realistic detectors have some level of noise; hence, Alice's and Bob's data will differ even in the absence of eavesdropping. Consequently, they must be capable of recovering from a reasonably low error frequency.

2. Producing a light pulse containing precisely one photon is technically challenging. It is far easier to generate a coherent pulse, which can be regarded as a superposition of quantum states with 0, 1, 2... photons; or an incoherent pulse, which can be seen as a statistical mixture of coherent states. In either case, let λ be the expected number of photons per pulse. If λ is small (i.e., significantly less than 1), there is approximately a 2/2 probability that an eavesdropper could split a pulse into two or more photons, reading one and allowing the other(s) to go to Bob. This enables the eavesdropper to acquire knowledge of a constant fraction of the bits shared between Alice and Bob without inducing errors.

A satisfactory protocol must be able to recover from both noise and partial leakage. Below, we outline a practical protocol that addresses these deficiencies, enabling Alice and Bob to reconcile the discrepancies between the sent and received versions of the quantum transmission. They can then distill from the reconciled data (about which the eavesdropper may possess significant partial information) a smaller set of data that is almost perfectly secret. The protocol we outline is simple but not optimal; other protocols that are currently under development have a higher yield of shared secret key at similar levels of noise and leakage.

Once the quantum transmission, Alice and Bob's first task is to exchange public messages to enable them to reconcile the discrepancies in their data. Since we assume that Eve listens to all public messages between Bob and Alice, this exchange must be conducted in a manner that reveals as little information as possible about this data. However, it's essential to remember that Eve cannot tamper with the content of these public messages.

A practical way for Alice and Bob to perform reconciliation is to initially agree on a random permutation of the bit positions in their strings (to randomize the error locations). They then partition the permuted strings into blocks of size k, with the belief that single blocks are unlikely to contain more than one error. (The optimal block size, which should be a function of the expected error rate, is yet to be theoretically determined. Instead, in Section 5, we use block sizes that have been empirically found to be effective.) For each such block, Alice and Bob compare the block's parity. Blocks with matching parity are provisionally accepted as correct, while those with discordant parity undergo a bisective search, revealing $\log(k)$ further parities of subblocks, until the error is identified and corrected. If the initial block size was significantly too large or too small due to an inaccurate a priori estimation of the error rate, this will become evident, and the procedure can be repeated with a more appropriate block size. To preserve privacy and prevent information leakage to Eve during the reconciliation process, Alice and Bob agree to discard the last bit of each block or subblock for which they have revealed the parity.

Of course, even with an appropriate block size, some errors will typically go undetected, occurring in blocks or subblocks with an even number of errors. To eliminate additional errors, the random permutation and block parity disclosure is repeated several more times, with increasing block sizes, until Alice and Bob estimate that at most a few errors remain in the data as a whole. At this point, the block parity disclosure approach becomes less efficient because it forces Alice and Bob to sacrifice at least one bit in each block for the sake of privacy. Consider, for example, a very common scenario where exactly two errors remain. If the block size is chosen such that there are I blocks, the probability of not detecting the remaining errors is 1/I, and the cost for this strategy is I bits when unsuccessful. Due to this, a different strategy is adopted to eliminate any errors that may remain and to verify, with high probability, that they have indeed been eliminated. The probability of undetected errors with this new strategy is 2^{-I} for the same cost of I bits sacrificed to privacy. Importantly, this probability is entirely independent of the number and location of the remaining errors.

In each iteration of this strategy, Alice and Bob compare the parities of a publicly chosen random subset of the bit positions in their entire respective data strings. If the data strings are not identical, then the parities of the randomly chosen subsets will disagree with a probability of exactly $\frac{1}{2}$. If a disagreement is detected,

Alice and Bob initiate a bisective search, akin to the one mentioned earlier, to identify and remove the error. As in the preceding block-parity stage of reconciliation, the last bit of each

compared subset is discarded to prevent any information leakage to Eve. Each subsequent random subset parity is computed using a new independent random subset of bit positions in the remaining string.

At a certain point, all errors will have been removed, but Alice and Bob will not yet be certain of their success. When this occurs, subsequent random subset parities will consistently agree. Following the last detected error, Alice and Bob continue comparing random subset parities until they find a sufficient number of consecutive agreements (say 20) to be confident that their strings are indeed identical, with a negligible probability of not detecting any remaining errors.

Alice and Bob now possess a string that is almost certainly shared, albeit only partially secret. As outlined in Section 4, they can derive a conservative estimate of Eve's partial information on their string from the detected error frequency and the optical pulse intensity. More precisely, they can estimate an integer l such that Eve's information about Alice's string resulting from the raw quantum transmission is worth no more than the knowledge of l physical bits of that string. Remember that the reconciliation process entails Alice disclosing the parity of numerous subsets of her bits, but with each disclosure, one bit from that subset is discarded from the reconciled string. As a result, Eve's knowledge about physical bits could transition into knowledge about parities. Let's say that Eve knows a parity bit about Alice's string if she is aware of the parity of a nonempty subset of the bits in that string (knowledge of physical bits is a special case of knowledge of parity bits, representing single elements of subsets). It's easy to observe that if Eve knows no more than l parity bits about a string y, and if she is given an additional parity bit about y, and z is formed by discarding from y one of the bits involved in that parity, then Eve still knows no more than l parity bits about z. Consequently, if Eve knew no more than l physical bits of Alice's string before reconciliation, she knows no more than l parity bits about the string shared between Alice and Bob that results from the reconciliation.

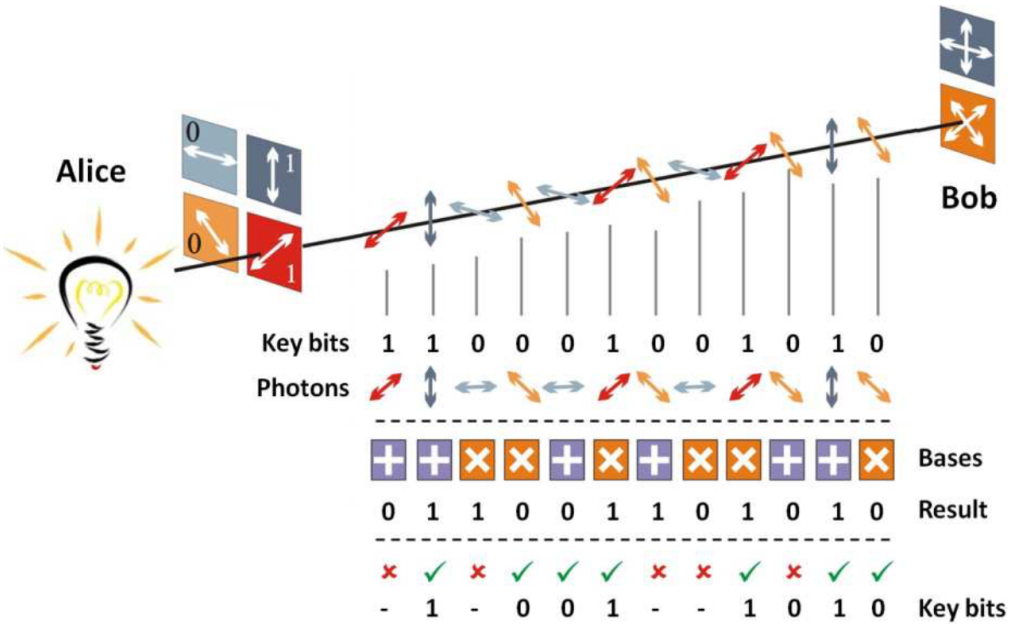


Figure 2.1. BB84 schema (source: [8]).

At this stage, Alice and Bob can proceed with privacy amplification. Let x represent the reconciled string, and n denote its length. We define a deterministic bit of information about x as the value e(x) of an arbitrary function e: $\{0,1\}^n \rightarrow \{0,1\}$. For example, physical and parity bits are deterministic bits, but bits of information in the sense of Shannon's information theory need not be. It's demonstrated in [9] that if Eve's knowledge about x is no more than l deterministic bits, a hash function h randomly and publicly chosen from an appropriate class

of functions $\{0,1\}^n \to \{0,1\}^{(n-t-s)}$ will map x into a value $h(x)$ about which Eve's expected information is less than $2^{-s}/\ln 2$ bit, where $s > 0$ is an arbitrary security parameter. This technique applies for Alice and Bob because parity bits are a special case of deterministic bits. An adequate hash function for this purpose can be obtained by continuing to compute $n - l - s$ additional publicly chosen independent random subset parities, but now keeping their values secret instead of comparing them. The class of hash functions thus realized is essentially the strongly-universal class H3 discussed by Wegman and Carter [7]. It's interesting to note that if even a single discrepancy is left between Alice's and Bob's data after reconciliation, the final strings computed by Alice and Bob will be completely uncorrelated, a fact likely to be noticed rapidly. Moreover, it's clear that this hash function has the property that if Eve's knowledge of x before privacy amplification was strictly in the form of parity bits, then such is also the case about her knowledge of $h(x)$. Therefore, Eve cannot have nonzero information about $h(x)$ without having at least one bit of information about it. Consequently, the privacy amplification theorem implies that Eve knows nothing at all about the final string $h(x)$ shared between Alice and Bob, except with a probability at most $2^{-s}/\ln 2$, in which case she knows at least one deterministic bit.

2.3 E91

E91 is a quantum key distribution protocol based on entanglement. The name of the protocol, E91, is a combination of the name of its inventor and the year of publication since it was proposed by Artur Ekert in 1991 [10]. The protocol is relatively simple, involving the following steps:

1. **Prepare Entangled Qubit Pairs** Qubit pairs are prepared in an entangled state.

2. **Send Qubits to Parties** Each qubit of the pairs is sent to the two parties involved in the communication.

3. **Measure Received Qubits** Both parties randomly choose bases for the measurements and measure the received qubits. Due to entanglement, if qubits are measured in the same basis, the results will be the same.

4. **Verify Entanglement** The parties engage in a public discussion to identify the qubits they measured with the same basis. For these qubits, they know the measurement outcomes are the same, which they keep to construct the key, discarding all other measurements.

The final step involves checking that the received qubits are genuinely entangled. This is done by calculating the correlation value of the received qubits and comparing it with the CHSH inequality. To ensure that the received qubits are in an entangled state, the correlation value (C) must satisfy the requirement: $\sqrt{C} \approx 2\sqrt{2}$ (Tsirelson's bound). This bound represents the maximal correlation value between two particles [11]. A correlation value approximately equal to this value provides proof that no interference occurred. If interference by an eavesdropper is detected, the correlation value will be in the range: $-2 \leq C \leq 2$, indicating interference.

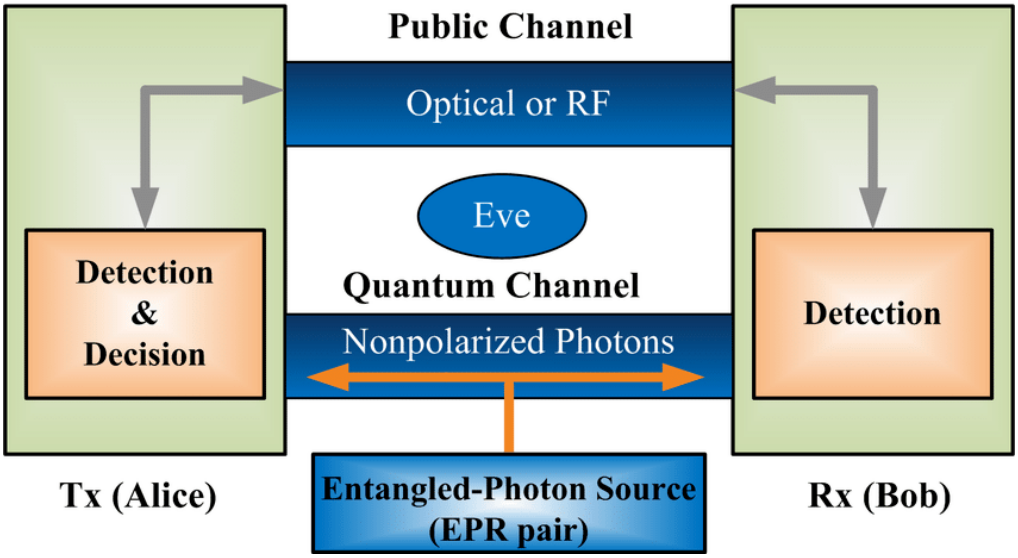


Figure 2.2. E91 schema (source: [12]).

A potential eavesdropper, Eve, can intercept both of the entangled qubits and measure them to obtain the results. However, since measurements destroy the qubit state, she will need to prepare two new qubits for Alice and Bob. She can encode in these qubits the measurement results she obtained. If Alice and Bob chose the same basis, all the parties will get the same results. However, the qubits sent by the eavesdropper are no longer entangled. The check on the correlation value will allow Alice and Bob to realize that someone interfered with the communication, leading them to discard the key.

Importantly, thanks to the properties of entanglement, it doesn't matter who prepares the entangled qubits. The source of qubits can even be an untrusted party or the eavesdropper itself without affecting the security of this protocol. This feature is known as Device Independent QKD (DI-QKD), where the security of the protocol is guaranteed even when untrusted QKD devices are employed [13].

2.4 QKD Post-Processing

Postprocessing involves a procedure where Alice and Bob refine the raw data obtained from quantum transmission into a secure key through public discussions. The process flow for QKD postprocessing is illustrated in Figure 2.3.

The concepts elucidated so far confirm that QKD has the potential to facilitate theoretically secure communication, aligning with the prerequisites of Shannon's theory. In an ideal scenario, QKD represents an optimal solution for the key exchange problem. However, in practical applications, errors and challenges must be taken into account.

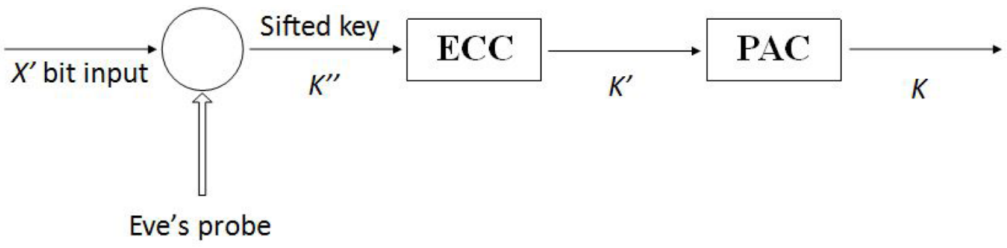


Figure 2.3. QKD schema (source: [14]').

Apart from errors that could be induced by the presence of a third party, actual devices used in transmission may introduce errors, similar to what happens in classical communication. Given that real devices are not flawless, polarization errors, detection inefficiencies, transmission losses, and external interferences (if the system is not adequately isolated) can modify the state of a quantum bit. It's imperative to consider these potential issues and devise appropriate mechanisms to detect and potentially rectify these errors.

To monitor any type of error that could arise during communication over a quantum channel, the Quantum Bit Error Rate (QBER) has been introduced. This parameter reflects the percentage of errors occurring during communication, encompassing both channel errors and those due to eavesdropping attempts. As every error on the channel potentially stems from a third party attempting to intercept the communication, efficiently estimating QBER becomes crucial to understand the extent of information leakage. Considering various error sources, QBER can be estimated as follows [15]:

$$QBER = p_f + \frac{p_d n q \Sigma f_r t_l}{2} \mu$$

Where:

- p_f: Probability of detector faults.
- p_d: Probability of a wrong photon signal.
- n: Number of detections.
- q: Phase = 12; polarization = 1.
- Σ: Detector efficiency.
- f_r: Pulse repeat frequency.
- t_l: Transmission rate (for large distances, small).
- μ: Attenuation for light pulses (single photons = 1).

If the resulting QBER is too high, the communication is deemed insecure, and the exchanged data should be discarded. QBER thus serves as an indicator of security for quantum communication. It's vital in this scenario to correctly identify the threshold for QBER: since channel errors are also included in QBER calculation, discarding communications surpassing a low threshold could result in discarding almost all communication. Conversely, selecting a high threshold could be risky, considering that potentially all errors could be attributed to information leaks.

2.4.1 Quantum Bit Error Rate (QBER) and Error Correction

In 2000, Shor and Preskill conducted a security analysis, determining the optimal QBER threshold for secure quantum communication [16]. They established that with a QBER of 11%, any potential eavesdropper cannot reconstruct the exchanged information, even when all errors are attributable to them. Consequently, it is a common practice in quantum communication to label information exchanged with a QBER above 11% as insecure.

The objective of information reconciliation is to ensure that Alice and Bob possess the same (raw) key [17]. Typically, this is achieved by considering Alice's bit string X as the key and letting Bob deduce this key from the information Y he possesses. In this process, Alice transmits partial information about X to Bob over the classical channel. The protocol, outlined in Figure 2.4, utilizes a raw key R and an authentic classical communication channel A. The goal is to generate a weak key resource R', which not only guarantees the secrecy of the key but also ensures that Alice and Bob's values, X and X', are identical.

```
protocol error correction (X,Y)
parameters enc, dec [coding scheme]
Alice sends C = enc(X) over the classical channel
Bob computes X' = dec(C, Y)
return (X,X')
```

Figure 2.4. Example Error Correction Protocol (source: [17])

It's important to note that information reconciliation is a purely classical subprotocol and is largely independent of other parts of the QKD protocol. The choice of the coding scheme, i.e., the functions *enc* and *dec* invoked by the protocol, depends on the noise model, describing how Alice and Bob's inputs X and Y are correlated. The noise model is typically specified by a joint probability distribution of X and Y. The coding scheme must be chosen to satisfy the condition:

$$p[dec(enc(X,Y)) = X] \geq 1 - \epsilon$$

where $\epsilon > 0$ bounds the failure probability of the subprotocol and contributes to the total failure probability of the QKD protocol. Moreover, to maintain maximum secrecy for X, the function *enc* should be designed such that $C = enc(X)$ does not reveal substantial information about X. This can be achieved by minimizing the size of C. Classical techniques from information theory demonstrate that any coding scheme satisfying eq.(2.1) requires a communication C of:

$$k \geq H^{\epsilon}_{max}(X|Y)$$

bits, where H^{ϵ}_{max} denotes the smooth max-entropy [18]. Specifically, for an i.i.d. noise model, when $QBER = \eta_0$, we have:

$$k \approx nh(\eta_0) + O(\sqrt{n})$$

Considering E as the initial information that Eve possesses about the raw key X before information reconciliation, the secrecy after information reconciliation with communication C consisting of k bits is given by:

$$H^{\epsilon}_{min}(X|EC) \gtrapprox H^{\epsilon}_{min}(X|E) - H_{max}(X|X') - O(\sqrt{n})$$

For the BB84 protocol, this leads to:

$$H^{\epsilon}_{min}(X|EC) \geq n(1 - 2h(\eta_0)) - O(\sqrt{n})$$

The amount of secrecy preserved after information reconciliation, as indicated by eq.(2.4), depends on the amount of communication k required. Designing coding schemes (enc, dec) that optimize this parameter is a key focus of classical information theory. While the bound in eq.(2.2) can be saturated with randomly constructed encoding functions, a primary challenge is to develop schemes for which the encoding and decoding functions are efficiently computable. While the information reconciliation protocol of Figure 2.4 involves one-way communication from Alice to Bob, two-way schemes are also considered. The cascade protocol is an example of a two-way scheme. In Chapter 3, we delve into the analysis of major error correction protocols.

2.4.2 Key Distillation and Privacy Enhancement

The purpose of privacy amplification is to transform the initially weakly secret key X [17], which, after information reconciliation, is known to both Alice and Bob, into a robust secret key K. This key K is a bit string that is nearly uniform and independent of any information that might be possessed by an adversary [19]. A typical protocol for this process is depicted in Fig. 2.5. Along with the weak key resource R (which meets a specified secrecy criterion and produces the identical string X for Alice and Bob), the protocol requires an authentic communication channel A. Utilizing these resources, the protocol constructs a secret key resource. It employs a randomness extractor [20], which is a family of functions ext_s parameterized by a seed $s \in S$. These functions take a bit string like X as input and generate a bit string of a fixed length l. In classical terms, a strong $(k, \epsilon) - extractor$ is defined such that for any input X with min-entropy satisfying $H_{min}(X) \geq k$, the output $ext_s(X)$ is ϵ-close to being uniform. Formally, the expectation over a randomly chosen seed $s \in S$ of the variational distance between the distribution of the output $ext(X)$ and a uniform string U of l bits must be upper bounded by ϵ:

$$Exp_s[D(P_{ext_s(X)}, P_U)] \leq \epsilon$$

However, this definition does not account for the quantum nature of information that an adversary may possess regarding X. Therefore, for application in quantum key distribution, especially when considering general security, it is imperative to demand that the randomness extractor $\{ext_s\}_{s \in S}$ be quantum-proof for parameters k and ϵ. This demands that, for any X and any quantum system E such that $H_{min}(X|E) \geq k$, the following holds:

$$Exp_s[D(\rho_{ext_s(X)E}, \rho_U \otimes \rho_E)] \leq \epsilon$$

This criterion refers to min-entropy with a smoothness parameter $\epsilon = 0$. However, a straightforward application of the triangle inequality for the distance between states implies that a similar criterion holds for the smooth min-entropy [18].

The most widely used extractors in the context of QKD are based on universal hashing [9]. These extractors can achieve an output length of $l = k - 2\log_2(1/\epsilon)$ while still being quantum-proof (k, ϵ) extractors. Employing them within the protocol of Fig. 2.5 generates a key of length:

$$l = H^{\epsilon}_{min}(X|EC) - O(1)$$

with a failure probability of the order ϵ. Combining this with the results of the previous sections, given optimal information reconciliation and privacy amplification, it's possible to generate a key of length:

$$l = H^{\epsilon}_{min}(X|E) - H_{max}(X|Y) - O(1)$$

In particular, for the BB84 protocol, this yields:

$$l = n(1 - 2h(\eta_0)) - O(\sqrt{n})$$

where η_0 is the QBER. The asymptotic key rate is therefore $1 - 2h(\eta_0)$. In Chapter 4, we will delve into the analysis of major privacy amplification protocols.

Chapter 3

Analysis of Error Correction protocols

3.1 QKD Error Correction

In the domain of quantum key distribution[21], errors within the initial keys exchanged between the communicating parties (Alice and Bob) can arise due to system noise or eavesdropping attempts.

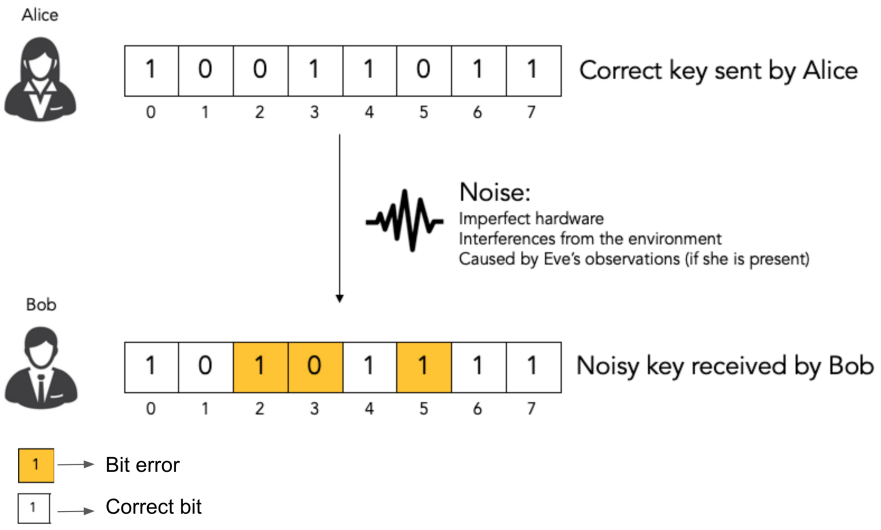


Figure 3.1. The key received by Bob contains some random errors (source: [22]).

These errors necessitate the reconciliation of keys by sharing additional information over a verified public channel. However, the information transmitted over this channel is also susceptible to eavesdropping by an external party (Eve), thereby compromising key privacy. Hence, the degree of information exposed serves as a yardstick for assessing the effectiveness of different reconciliation protocols.

The earliest protocol, known as BBBSS [23], was introduced by Bennett et al. in 1992. In this method, Alice and Bob segment their key strings into blocks and exchange the parity of each block. When they identify a block with varying parity, they employ binary search within the block to rectify an error. This process involves multiple iterations, and between these iterations, the key strings are randomly rearranged.

The subsequent Cascade protocol [24], presented in 1993, builds upon BBBSS. Cascade retains records of the blocks during each iteration. When a new error surfaces, it's possible to trace back to historical records to identify corresponding errors. This adaptation reduces the volume of disclosed information, thereby enhancing protocol efficiency. Both BBBSS and Cascade, however, demand frequent communication between Alice and Bob, which could impede the process in practical applications.

Another class of protocols requires notably less communication. In these scenarios, key strings are again divided into blocks, but rather than performing binary search, a syndrome is transmitted. This syndrome is computed based on a specific error-correcting code.

An exemplar of this kind is the Winnow protocol [25]. In Winnow, a syndrome based on a Hamming code is utilized. When Alice and Bob detect a block with differing parity, Alice sends Bob the syndrome of the block. Bob then endeavors to rectify his own block using the syndrome, but this process might introduce new errors. Consequently, Winnow offers higher speed but demands the exchange of more information.

In terrestrial links, time, computation, and communication complexity are typically less restrictive for Alice and Bob. However, satellite links introduce challenges due to significant channel losses, limited windows for key establishment due to periodic satellite passages, and the additional constraints posed by computation and communication complexities.

In [26] during recent years, researchers have turned to Gallager's Low Density Parity Check (LDPC)[27] codes, which have exhibited superior error reconciliation rates compared to Cascade and Winnow[28]. LDPC codes present low communication overhead and inherent computation power asymmetry between the communicating sides.

LDPC linear codes rely on a parity check matrix H and a generator matrix, defining a code's decoding limit through its minimum distance. The predefined code rate determines its correction power and efficiency

Decoding LDPC codes entails larger computational and memory demands than Cascade or Winnow algorithms. Yet, it offers a notable advantage in terms of reduced communication resources, requiring only one information exchange. In scenarios with restricted resources (such as bandwidth and latency), this trade-off can yield significant improvements in overall runtime and confidentiality. Within the realm of QKD, LDPC was initially adopted as the foundation for the BBN Niagara protocol in the DARPA QKD network [29].

In this chapter we will present some of the evaluation metrics for error correction protocols and we will analyze the main ones.

3.1.1 EC metrics

Preliminaries

In [24] the reconciliation efficiency metric was proposed as follows.

Consider a probability distribution $\{P(x)\}_{x\in X}$ over a finite set X. The entropy of this distribution applied to X, denoted as $H(X)$, is mathematically defined as

$$H(X) = -\sum_{x\in X} p(x)\log p(x)$$

(assuming all logarithms are base 2)[24]. Essentially, $H(X)$ represents the average number of bits needed to describe a specific event in X. It's worth noting that $H(X)$ has an upper bound of $\log|X|$:

$$H(X) \leq \log|X|$$

Equality is only achieved when every element x in X has a probability $p(x)$ equal to $1/|X|$. When dealing with a Bernoulli trial having a parameter p, $H(X)$ is denoted as $h(p)$. Consider

two sets, X and Y, with a joint probability distribution $\{p(x,y)\}_{x\in X, y\in Y}$. In this context, the conditional entropy $H(X|Y)$ is defined as

$$H(X|Y) = -\sum_{y\in Y}\sum_{x\in X} p(y)p(x|y)\log p(x|y).$$

A binary symmetric channel (BSC) facilitates the transmission of a sequence of bits, each independently exposed to noise with a probability of p. Assume A as the sequence transmitted by Alice and B as the sequence received by Bob. When each bit in the sequence A is randomly and independently selected, it's evident that

$$H(A) = |A|$$

The conditional entropy of A given B is given by

$$H(A|B) = H(A \oplus B) = nh(p)$$

where $n = |A| = |B|$. From now on, a binary symmetric channel having the parameter p is referred to as $BSC(p)$. The quantum channel serves as an instance of a concealed binary symmetric channel, even when an eavesdropper introduces noise asymmetrically. Prior to aligning their sequences, Alice and Bob sample the transmitted bits to estimate the error rate in the quantum communication. If this estimate is sufficiently close to the expected error rate of the channel, they openly and randomly shuffle their respective sequences and then apply reconciliation. Subsequently, Bob's sequence can be treated as the outcome of a transmission over a BSC. However, if the estimate doesn't closely match the anticipated error rate, they discard the communication attempt and try again later. For values $0 \leq \lambda \leq \frac{1}{2}$, the tail inequality is given by

$$\sum_{k=0}^{[\lambda n]} \binom{n}{k} \leq 2^{nh(\lambda)}$$

When dealing with a random variable X having a finite variance $V(X)$ and an expected value $E(X)$, along with a positive value a, the Chebyshev inequality can be expressed as

$$prob(|X - E(X)| \geq a) \leq \frac{Var(X)}{a^2}$$

The Hamming distance [30] $dist(A,B)$ between sequences A and B is defined as the count of positions where A and B differ. Additionally, the weight $w(A)$ of sequence A is the number of its non-zero positions.

Reconciliation Efficiency

In [31] the reconciliation efficiency is formalized as follows using the same format that will be used for this discussion. Consider two correlated discrete random variables, denoted as X and Y, both having a binary alphabet $A = \{0,1\}$. Their joint probability is given by $pXY(x,y) = Pr(X = x, Y = y)$. For clarity, we will omit the random variables when it doesn't cause confusion. The probability $p(x,y)$ can also be represented as $p(y|x)p(x)$. Here, y can be seen as the result of a memoryless channel characterized by the transition probability $p(y|x)$ with input x. In the context of discrete-variable Quantum Key Distribution (QKD), the discrepancies between variables x and y from two distant parties, Alice and Bob, respectively, are assumed to occur due to transmission over a binary symmetric channel with crossover probability ϵ, denoted as $BSC(\epsilon)$. This parameter ϵ is often referred to as the Quantum Bit Error Rate (QBER).

Let $x \in A^n$ and $y \in A^n$ be the outcomes of n independent and identically distributed (i.i.d.) instances of X and Y, respectively. For clarity moving forward, we'll call these sequences "frames".

The reconciliation problem can be viewed as a specific case of source coding with side information, which is also known as Slepian-Wolf coding [32]. When a source X is paired with a decoder having access to side information Y, no encoding of X shorter than $H(X|Y)$ guarantees reliable decoding by the receiver [32]. Therefore, the minimum information needed is represented by the conditional entropy $H(X|Y)$. Let m be the length of the exchanged message to reconcile the disparities between x and y. In this context, the efficiency of an information reconciliation procedure can be quantified by the formula:

$$f_{EC} = \frac{m}{nH(X|Y)}.$$

Since $nH(X|Y)$ represents the minimum required message length to reconcile the frames x and y, it follows that $f_{EC} \geq 1$, and $f_{EC} = 1$ indicates perfect reconciliation.

In the case of a $BSC(\epsilon)$, the reconciliation efficiency can be expressed as:

$$f_{EC} = \frac{1 - R}{h(\epsilon)}$$

where the binary Shannon entropy $h(\epsilon) = -\epsilon \log_2 \epsilon - (1 - \epsilon) \log_2(1 - \epsilon)$, and R represents the ratio of transmitted information, specifically $R = 1 - m/n$. The difference $1 - R$ signifies the proportion of redundant information disclosed to reconcile errors.

It's important to note that the reconciliation efficiency is often interpreted differently in various literature. While we've defined it as a measure of the additional information disclosed beyond the Shannon limit, other works define it as the ratio of achieved capacity for a given communication channel. In this alternative interpretation, the efficiency β is given by:

$$\beta = \frac{R}{1 - h(\epsilon)}$$

resulting in the relationship:

$$1 - f_{EC} h(\epsilon) = \beta(1 - h(\epsilon))$$

Throughout this discussion, we exclusively adopt the initial definition.

Execution time

When it comes to assessing the performance of error correction algorithms, the metric of execution time assumes a crucial role. This holds true even within the realm of quantum communication, where intricate processes like privacy amplification are employed. Despite the fact that privacy amplification, a process critical to refining partially secure raw keys into robust secret keys, can be computationally demanding, the evaluation of error correction techniques still factors in execution time.

In quantum communication, especially Quantum Key Distribution (QKD), the focus on execution time underscores the practicality and feasibility of error correction algorithms. While privacy amplification may present computational bottlenecks in the post-processing stages of QKD, evaluating error correction algorithms using execution time remains pertinent. This is because the efficiency and effectiveness of these algorithms can directly impact the overall reliability and security of the communication process.

While privacy amplification itself may involve intricate computations, the consideration of execution time is not diminished. It's an essential facet in the larger landscape of error correction, ensuring that algorithms not only deliver accurate corrections but do so within time frames that align with the operational requirements of the communication system.

In essence, the evaluation of error correction algorithms goes beyond their theoretical capabilities. It extends into the realm of real-world applicability, where execution time plays a pivotal role in determining their practical utility. In the dynamic context of quantum communication, this evaluation balance becomes even more important, ensuring that both correction accuracy and operational speed are harmonized for effective and efficient data transmission.

3.2 Winnow

In [25] the Winnow protocol was presented as follows, we now report it in order to compare it to the other main EC protocols.

3.2.1 Hamming Error Detection and Correction

Let's elucidate the practical application of the Hamming hash function in the realm of error correction [30]. The process is as follows:

1. To begin, following the exchange of qubits between A and B via the quantum channel, A and B proceed to partition their random bits into distinct blocks. These blocks are characterized by a length denoted as $N_h = 2m - 1$. Given the inherent one-to-one correspondence within this dataset, these blocks are henceforth referred to as individual data units or bit-blocks. It's essential to highlight that the value of m must meet or exceed 3.

2. Subsequently, A and B independently embark on computing m-bit syndromes, labeled as S_a and S_b respectively. These syndromes are contingent solely upon the bits within a particular block, whether from A or B.

3. The next step involves B transmitting his syndrome to A. It's imperative to note that errors come to light only when a syndrome difference, denoted as S_d, is evident. This syndrome difference is computed as the result of an exclusive OR operation between S_a and S_b:

$$S_d = S_a \oplus S_b \neq \{0\}^m.$$

4. Finally, to mitigate the risk of privacy breaches, which may arise due to the classical communication of B's syndromes to an entity represented as E, m bits are systematically removed from each bit block. This action results in the revelation of m bits of information in each block where S_b is exposed. Consequently, this leads to a reduction in the channel's capacity per symbol, a factor of m/N_h [10].

The preservation of data privacy is achieved by eliminating specific bits from each block, targeting positions corresponding to $\{2^j\}$, where $j \in \{0, ..., m-1\}$. These excised bits maintain independence during subsequent syndrome calculations, as evidenced by the matrix $h^{(m)}$:

$$h^{(3)} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

In this specific instance, m is denoted as 3. The process of removing bits in this fashion is referred to as privacy maintenance [4].

As a concluding remark regarding the aforementioned matrix, it's noteworthy that the transpose of $h^{(3)}$, represented as $[h^{(3)}]^T$, corresponds to binary equivalent numbers ranging from 1 to 7. This generalizes to the statement that $[h^{(m)})]^T \equiv 1, ..., (2m-1), Nh$ binary numbers.

This matrix holds a distinct position as a specialized form of a hash function [11] and is represented as:

$$h_i^{(m)} = [\frac{j}{2^{i-1}}](mod 2)$$

Within this equation, $i \in 1, ..., m$, and $j \in 1, ..., N_h$, and the arithmetic operations are performed modulo 2.

Looking ahead, it's important to note that the Hamming algorithm excels in rectifying any single error within a N_h-bit block. Nevertheless, the extent of its efficacy, particularly concerning syndromes and privacy maintenance, becomes less evident when confronted with scenarios involving multiple errors within a single bit block. These specific considerations will be meticulously examined as we delve into the complexities surrounding syndromes.

The syndromes S_a and S_b are formed by contraction of the N_h-bit blocks with the matrix $h^{(m)}$:

$$S_i = \left(\sum_{j=1}^{N_h} X_j h_{i,j}^{(m)} \right) (mod 2) \in \{0,1\}^m$$

where subscript i represents syndrome bit i in the m-bit binary syndrome, X_j represents bit j in$A$'s or$B$'s block, and $S = S_i$ is the binary syndrome value of either B's or A's block. To comprehend the Impact of Syndromes on Error Identification and Correction it is imperative to grasp how syndromes play a pivotal role in the localization and rectification of errors. This comprehension forms the bedrock for evaluating the efficacy of the Hamming code and, by extension, the Winnow algorithm.

The concept of a "syndrome difference" delineates a binary value that specifies the position of a single bit within the code word of either A or B. This particular bit, when toggled from 0 to 1 or vice versa, exerts an influence on the syndrome difference denoted as S_d. Subsequently, upon recalculating the syndrome difference, it converges to a binary value $S'_d \equiv 0$.

For instance, when $S_d \neq 0$, it signifies that S_d is a binary value consisting of m bits. The value of this binary representation indicates the precise location of a single bit within the code word of either A or B. This bit is then subjected to an exclusive OR operation with its original value. After this modification, a fresh syndrome for that specific code word is computed, denoted as S'_A. This newly computed syndrome is again subjected to an exclusive OR operation, this time with the original syndrome of the other code word, represented as S_B in this example.

The outcome of this intricate process is the alteration of the single bit indicated by the non-zero syndrome difference within the code word. This can lead to one of two scenarios: either the error is corrected, or a new error is introduced within that code word. This process is not shrouded in mystery but rather is a direct consequence of the fact that Hamming codes belong to the class of n-k codes.

In this context, where $n = 2m - 1$, n relates to the number of bits within each code word (denoted as N_h), while $k = n - m$ pertains to the channel's capacity. The relationship between k and n signifies the channel capacity per bit, expressed as $k/n \iff k/Nh$. This relationship is intrinsic to the specific code under consideration, which, in this discussion, is a Hamming code.

Within an n-k Hamming code, there exist $2^{(2^m)}$ distinct code words, each distinguished by 2^m unique syndromes. Furthermore, within this code, there are also 2^k code words that share identical syndromes. Due to the code's capability to rectify a solitary error, it exhibits a minimum Hamming distance of $d = 3$. This characteristic also implies its ability to detect a minimum of two errors. Indeed, the Hamming distance, denoted as d, for the Hamming code is invariantly equal to 3.

By definition, a code word hosting a solitary error exhibits a syndrome difference denoted as $S_d \neq \{0\}^m$ - a clear indication of its capability to detect a single error, aligning with its error correction capacity. Furthermore, for a code word harboring precisely two errors, it follows by definition that $S_d \neq \{0\}^m$. Thus, a code word can discern a minimum of two errors if it can rectify a single error. Consequently, when a code word with exactly two errors undergoes the Hamming algorithm, and the bit specified by S_d is altered, it ultimately concludes with precisely three errors.

The proof of this assertion is rooted in contradiction: if a code word with two errors concluded with only one error (signifying an error correction), the new syndrome difference would be non-zero - a contradiction. This line of reasoning also firmly establishes that one error is corrected when there is precisely one error. If an error were introduced, the syndrome difference would once again be non-zero.

Hence, when scrutinizing Hamming codes, it becomes evident that a code word initially plagued by one error eventually reaches a state of error-free, while a code word initially bearing precisely two errors ultimately acquires three errors. In both cases, the new syndrome difference metamorphoses to $S'_d = \{0\}^m$.

Expanding this understanding, by symmetry, if an N_h-bit code word contains precisely h errors (with all bits except one being erroneous), the application of Hamming ensures that all bits in the code word will be in error. Conversely, a code word initially burdened with N_h - 2 errors will conclude with N_h - 3 errors, implying the correction of one of the errors.

These deductions bring to light a crucial aspect of Hamming codes: their efficiency depends on the probability of encountering two or more errors compared to the likelihood of single or zero errors. In either of the latter cases, the Hamming code is inefficient since it divulges m-bits within the syndrome. We will delve deeper into this aspect later.

The complexity arises when attempting to analyze how Hamming behaves in scenarios involving more than two but fewer than 2^{m-1} errors within code words. It's not immediately evident how the number of code words with three errors and $S_d \equiv \{0\}^m$ correlates with the number of ways two-error code words transform into code words with three errors, accompanied by $S_d = \{0\}^m$. In essence, it suggests that there must exist a method to arrange three errors within a code word while maintaining $S_d = \{0\}^m$. Failure to identify this mechanism would imply that the code could unfailingly detect more than two errors, which would contradict the fundamental Hamming distance of $d = 3$.

To comprehensively evaluate Hamming's efficiency, we must scrutinize the behavior of code words afflicted with three or more errors following the application of Hamming. Let's start with the case of three errors, which is now evidently clear: there must exist a minimum of 2^{m-1} approaches to begin with three errors in an N_h-bit code word and ultimately maintain three errors.

In scenarios where three errors exist within a code word, and $S_d \neq \{0\}^m$, introducing an error into the N_h-bit code word is inevitable. This stems from the fact that if the code word were to conclude with two errors, S_d would not equate to $\{0\}^m$, which contradicts our assumptions.

To illustrate this, let's consider a specific example with $m = 3$. There are $\binom{7}{3} = 35$ ways to arrange three errors in seven bits. As there are precisely seven non-zero syndrome differences for $m = 3$, with $n_i = 2$, there must be a minimum of seven methods to arrange three errors within seven bits and have $S_d \equiv \{0\}$. In this specific case, this is indeed the result. This statistical analysis implies that, among 35 code words with three errors, approximately 7 out of 35 will remain with three errors, while the remaining 28 out of 35 words will transform into code words with four errors. Consequently, code words initially plagued by three errors will conclude with 19/5 errors per 7-bit block, assuming an infinite number of 7-bit blocks each initially containing exactly three errors.

Applying symmetry, it becomes evident that in a scenario featuring an infinite number of 7-bit blocks each initially possessing four errors, the final error rate per block would be lower at 16/5.

Hence, what becomes essential is a methodology to calculate, for any given value of m representing parity checks in Hamming, the number of ways to arrange the initial error count per block while ensuring the final syndrome results in $S = \{0\}^m$, or with $S_d \neq \{0\}$. The following equations permit that calculation for any initial number of errors per block, n_i, given any initial block size, N_h:

$$N_{S_{d\neq 0}} + N_{S_{d=0}} = \binom{N_h}{n_i} - N_{S_{d\neq 0}} + N_h \cdot N_{S_{d=0}} = (-1)^q \cdot N_h \cdot \binom{\frac{N_h-1}{2}}{p} \iff$$
$$\iff \begin{bmatrix} N_{S_d\neq 0} \\ N_{S_{d=0}} \end{bmatrix} = \begin{bmatrix} N_h & 1 \\ -1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} \binom{N_h}{n_i} \\ (-1)^q \binom{\frac{N_h-1}{2}}{p} \end{bmatrix}$$

where $q = \lfloor \frac{n_i}{2} \rfloor$, $p = \lfloor \frac{n_i}{2} \rfloor$, and n_i signifies the initial error count per Hamming block consisting of $N_h = 2^m - 1$ bits, two crucial metrics come into play. These are $N_{S_{d=0}}$, representing the count of syndrome differences yielding $S_d = 0^m$, and $N_{S_{d\neq 0}}$, denoting the count of syndrome differences with $S_d \neq 0^m$.

The subsequent equations provide a broader perspective by normalizing both sides with the total number of ways to arrange n_i errors among the N_h bits. This leads us to more useful quantities:

$$\prod_{S_{d=0}} = \frac{N_{S_{d=0}}}{\binom{N_h}{n_i}}, \text{ and}$$
$$\prod_{S_{d\neq 0}} = \frac{N_{S_{d\neq 0}}}{\binom{N_h}{n_i}}$$

This result will prove valuable in subsequent discussions. While these arguments might not directly apply to cases where $m > 3$, they provide insights into the general problem. The challenge with the specific scenario of $m = 3$ and $n_i = 3$ is that the next case, where $n_i = 4$, is symmetric and complementary to $n_i = 3$, as previously mentioned. Moreover, as noted earlier, there's no pathway to transform 3 errors into 2 errors since $S_d \neq 0^m$ when exactly 2 errors exist. Nevertheless, the aforementioned equation serves as a general technique to calculate these quantities. It essentially provides the number of ways to transform n_i errors into $S_d = 0^m$ or not, considering $N_h = 2^m - 1$ bits in a block.

Now, turning our attention to scenarios where $m \geq 4$ and $4 \leq n_i < 2^{(m-1)}$, the key question centers on how errors evolve. Here, let's denote by $n(+)$ the number of ways to increase the error count from n_i to $n_i + 1$ within a bit-block, and n_i the number of ways to decrease the error count from n_i to $n_i - 1$. Naturally, these considerations apply when $m \geq 4$. The outcomes are as follows:

$$n_i^{(+)} = N_{S_{d=0}}(N_h|n_i) + (n_i + 1) \cdot N_{S_{d=0}}(N_h|n_i + 1)$$
$$n_i^{(-)} = \binom{N_h}{n_i} - n_i^{(+)}$$

In this context, $NSd = 0(N_h|n_i+1)$ represents the count of ways to distribute n_i+1 errors among N_h bits and yield $S_d = \{0\}^m$. As mentioned earlier, it's worth noting that the count of ways to achieve $S_d = \{0\}^m$ for $n_i + 1$ errors is directly linked to the count of ways to transition from n_i errors to $n_i + 1$ errors. Naturally, $N_{S_{d=0}}(N_h|n_i)$ corresponds to the count of ways to distribute n_i errors among N_h bits to obtain $S_d = \{0\}^m$. Consequently, the generalized probability of increasing or decreasing the error count, denoted as n_i, can be expressed as follows:

$$\overset{(+)}{\prod} = \frac{n_i^{(+)}}{n_i^{(+)} + n_i^{(-)}}, and$$
$$\overset{(-)}{\prod} = 1 - \overset{(+)}{\prod}.$$

3.2.2 Winnowing

In the quest for an ideal error correction protocol, the goal is to rectify all bit errors within each data block, refrain from introducing additional bit errors, and disclose as little crucial information as possible during public communication to safeguard against eavesdropping. However, the Hamming protocol, as outlined, falls short of this ideal in several aspects.

Firstly, the syndrome difference S_d fails to distinguish between single-bit errors and multiple-bit errors. Consequently, treating instances of $S_d \neq 0^m$ as single errors may inadvertently introduce more errors into the system.

Secondly, during each exchange, up to m bits of information are swapped for every data block. This exchange diminishes the channel's capacity per symbol and is vulnerable to eavesdropping, posing a security risk.

One potential solution involves eliminating all bits within data blocks where $S_d \neq 0^m$. While this approach eliminates the risk of introducing additional bit errors into the key, it is not highly efficient. It results in the loss of either m bits per block due to privacy maintenance or, in the worst-case scenario, all bits if $S_d \neq 0^m$. The drawback here is that most of the discarded bits or blocks with $S_d \neq 0^m$ are likely not erroneous.

An alternative and more robust solution is to introduce an initial parity comparison on a block comprising $N = 2^m$ bits. This comparison allows for an assessment of the syndromes S_a and S_b based on the outcome of the parity comparison, thus offering a more nuanced approach to error correction.

If the block parities do not align, it signifies an odd number of errors in the N-bit block. Furthermore, when the bit errors are randomly distributed throughout the data and their count is relatively small, an odd number of errors in a block likely indicates a single error. This single error can be corrected by subsequently applying the Hamming algorithm.

For instance, consider a scenario where a block contains one bit error. If $S_d = 0^m$, it implies that the first bit is in error. It's worth noting that by symmetry, if there are precisely $N-1$ errors in the block, the first bit wouldn't be in error. This approach, therefore, consistently enables the correction of a single error among the N bits, assuming the bits are to be retained.

However, in the protocol described here, one bit is routinely discarded for privacy maintenance (pertaining to the exchanged parity bit). The Hamming algorithm is then applied to the remaining N_h bits, as discussed earlier. Following this, an additional $[\log_2(N_h)]$ bits are discarded to finalize privacy maintenance. This results in a channel capacity of $(2^m - m - 1)/N$ per symbol for blocks containing an initial parity error. While it might seem like an additional loss of channel capacity, the absence of syndrome exchange and comparison when the block parities align actually increases the channel capacity over the basic Hamming algorithm. For blocks that don't exhibit a parity error, one bit is still discarded for privacy maintenance.

This error reconciliation protocol is referred to as Winnow. In the event that the parities on the N bits don't agree, Winnow discloses $log_2(N) + 1$ bits through two classical communications: m bits for the syndrome and 1 bit for parity. Conversely, when the parities align, Winnow reveals 1 bit of information in a single classical communication. Therefore, the amount of key data that gets discarded is:

$$N_{dis.}^{odd} = \log_2(N) + 1 = m + 1$$

bits for blocks with odd numbers of errors such that the fraction of the bits remaining after privacy maintenance is

$$\mu_{pm}^{odd} = 1 - \frac{N_{dis.}^{odd}}{N}$$

For $N \in 8, 16, 32, 64, 128 \mu_{pm}^{odd} \in \{0.5, 0.69, 0.88, 0.89, 0.94\}$, respectively. Also,

$$\mu_{pm}^{even} = 1 - \frac{1}{N}$$

For the same values of N, μ_{pm}^{even} takes on values within the set $0.88, 0.94, 0.97, 0.98, 0.99$. In either scenario, the appropriate overhead for classical communications is immediately subtracted from the data. This ensures that the privacy of the bits is at least maintained, if not improved.

After a single pass of Winnow (a Winnowing), all single bit errors in an N-block are guaranteed to be either eliminated or corrected. What remains to be examined is how blocks with multiple errors impact the overall efficiency of Winnow.

3.3 Low Density Parity Check

LDPC codes implementation for QKD was presented first in [29] and present a different paradigm of QKD error correction which focuses on pre-computation of LDPC codes instead of using iteratively a classical channel to achieve error correction. In [33] has been revisited as follows

3.3.1 Error correction and control coding

Low-Density Parity-Check (LDPC) codes, possessing a sparse parity-check matrix, are linear error-correcting codes pivotal in approaching the Shannon capacity limit for performance. The defining characteristic of an LDPC code lies in its parity-check matrix denoted by H. Furthermore, LDPC codes can be accurately portrayed using bipartite graphs closely associated with their parity-check matrices H. These graphs encompass check nodes representing vertices for parity-check equations, and variable nodes (or bit nodes) embodying vertices for the codeword bits. The interconnection of variable nodes and check nodes by edges forms the basis of the bipartite graph and is fundamental to the message-passing decoding algorithm. LDPC codes can be categorized into two main types: regular LDPC codes, where all variable nodes and check nodes possess identical degrees, and irregular LDPC codes, where degrees can vary. The degree distribution of irregular LDPC codes can be characterized using degree distribution polynomials $\lambda(x)$ and $\rho(x)$:

$$\lambda(x) := \sum_{i=2}^{d_v} \lambda_i x^{i-1}$$

$$\rho(x) := \sum_{i=2}^{d_c} \rho_i x^{i-1}$$

Here, λ_i represents the fraction of edges in the bipartite graph connected to variable nodes of degree i, and ρ_i represents the fraction of edges connected to check nodes of degree i. Moreover, $\sum_i \lambda_i = \sum_i \rho_i = 1$, while d_v and d_c denote the maximum degree for variable nodes and check nodes, respectively. The degree distribution pair is instrumental in predicting the decoding threshold for LDPC codes [34].

For a given binary sequence $X = (x_1 x_2 \dots x_n)$ of length n, the syndrome bits S are computed as an encoder output by performing a modulo-2 operation on the product of the source bits X and the parity-check matrix H, as expressed by

$$S = mod\{H * X^T, 2\}$$

The syndrome $S = (s_1 s_2 \dots s_{n-k})$, with $s_j \in \{0,1\}$, represents the value of the j_{th} syndrome component. Equivalently, in the bipartite graph, this operation can be seen as binary addition of all the variable node values connected to the same check node. The matrix H is an $(n-k) \times n$ LDPC parity-check matrix. Due to the sparse nature of H, the encoding complexity of LDPC codes for traditional error correction applications is proportional to the square of the length of the codeword. The encoded version of X is the syndrome S, which serves as the input to the authenticated classical channel. The source Y is available at the decoder as side information. Utilizing a linear (n, k) binary block code, it becomes possible to generate 2^{n-k} distinct syndromes, each indexing a set of 2^k binary words of length n [35]. This encoding process maps a sequence of n input symbols into $(n-k)$ syndrome symbols.

Given the side information Y and the received syndrome S, the objective of the decoder is to reconstruct the best estimate X of the source n-sequence X. The fundamental LDPC decoding algorithm encompasses the bit-flipping algorithm and the sum-product algorithm. The bit-flipping algorithm is grounded on the principle that a code word bit involved in a substantial number of incorrect check equations is likely to be incorrect itself. Thus, if the majority of the messages received by a bit node differ from its received value, the bit node changes (flips) its current value. Iterative decoding of binary LDPC codes using the sum-product algorithm (SPA)

represents a soft decision message-passing algorithm. The input bit probabilities are denoted as a priori probabilities for the received bits, and the bit probabilities returned by the decoder are termed a posteriori probabilities. In the case of sum-product decoding, these probabilities are expressed as log-likelihood ratios (LLR). The steps involved in information reconciliation for Quantum Key Distribution (QKD) are akin to the classical belief-propagation algorithm, with a modification introduced at step 2. The LLR of a binary random variable x is defined as:

$$L(x) = \log_e \frac{p(x=0)}{p(x=1)}$$

where $p(x=1)$ denotes the probability that the random variable x takes the value 1, and

$$p(x=1) = \frac{\frac{p(x=1)}{p(x=0)}}{1+\frac{p(x=0)}{p(x=1)}} = \frac{e^{-L(x)}}{1+e^{-L(x)}}$$

For a binary variable x, it is straightforward to find $p(x=0)$ given $p(x=1)$, since $p(x=1) = 1-p(x=0)$. The LLR-SPA can be summarized as follows:

Step 1: Initialization: Set the log-likelihood ratios p to appropriate initial values based on the knowledge of the mean source correlation, i.e.,

$$M_{j,i} = r_i$$

where M_j, i is the messages sent from the bit nodes to the check modes, i = 1,2, ..., n, j = 1,2, ..., $n-k$, r_i is the log likelihood ratios for the priori message probabilities in BSC, and

$$r_i = \begin{cases} \log_e \frac{p}{1-p}, \ if \ y_i = 1 \\ \log_e \frac{1-p}{p}, \ if \ y_i = 0 \end{cases}$$

Step 2: Calculate $E_{j,i}$: the extrinsic message from check node j to bit node i, defined as the LLR of the probability that bit i causes parity-check j to be satisfied.

$$E_{j,i} = \log_e\left(\frac{1+(1-2s_j)\prod_{i' \in B_j, i' \neq i} \tanh(M_{j,i'}/2)}{1-(1-2s_j)\prod_{i' \in B_j, i' \neq i} \tanh(M_{j,i'}/2)}\right)$$

where B_j is the set of bits in the j^{th} parity-check equation of the code. s_j is the j^{th} syndrome component sent from Alice. This equation represents the only modification to the classical belief-propagation algorithm, where information is propagated between variable and check nodes connected by edges.

Step 3: Calculate the total LLR of the i^{th} bit: given by the sum of these LLRs:

$$L_i = \sum_{j \in A_i} E_{j,i} + r_i, \ i = 1,2, ..., n$$

where A_i is the parity-check equations that check the i^{th} bit of the code.

Step 4: Hard decision on the received bits: given by the sign of the LLRs.

$$z_i = \begin{cases} 1, \ if L_i \leq 0 \\ 0, \ if L_i > 0 \end{cases}$$

Step 5: Check if $Z = (z_1, z_2 ..., z_n)$ is a valid codeword:

$$S' = mod\{H * Z^T, 2\}$$

Then Bob compares the syndrome S' with the syndrome S sent from Alice. If $S' \neq S$, then there are errors in Y, and the process goes back to step 2; the belief propagation of LDPC decoding is iteratively computed until $S' = S$, or the maximum allowed iteration number of decoding is reached. The result $S' = S$ signifies that the sifted key is identical in Alice's and Bob's after information reconciliation; otherwise, there are still some erroneous bits between Alice's and Bob's sifted keys. These erroneous bits can no longer be corrected in one-way reconciliation protocols.

3.3.2 Optimization Work

In [36] further work on LDPC for QKD was carried, improving performance exploiting Differential Evolution.

Low-Density Parity-Check (LDPC) codes, also known as Gallager codes, represent linear codes characterized by a sparse parity check matrix with relatively few non-zero values. Their significant advantage lies in their ability to perform very close to the Shannon limit, even with suboptimal yet fast, iterative decoding schemes. In the context of reconciliation of binary strings, particularly for application to discrete-variable Quantum Key Distribution (QKD), LDPC codes need to be specially optimized for the Binary Symmetric Channel (BSC). The optimization problem of LDPC code design can be effectively addressed using a genetic algorithm, specifically Differential Evolution (DE) [37]. This solution has been successfully applied to the Binary Erasure Channel (BEC) [38] and the Binary Input Additive White Gaussian Noise (BIAWGN) channel [39].

Differential Evolution (DE) is an Evolutionary Optimization Algorithm that maintains a population of N D-dimensional vectors (code candidates) of real parameters, adhering to certain constraints. The population evolves for a fixed number of generations or until a vector is found that meets a stopping criterion. The population is initialized to cover as much of the parameter space as possible. For each generation, DE mutates and recombines the current population to produce a trial population. Mutation is performed by adding the weighted difference of two population vectors to a third one. Recombination is used to increase the diversity of the trial population, where trial vectors are modified incorporating a small set of parameter values from a current population vector. A trial vector is incorporated into the current population if a cost function assigns to it a lower cost value than the cost value of the preceding vector; otherwise, it is discarded.

LDPC codes can be represented as bipartite graphs, where one set of nodes, the check nodes, represents the set of parity-check equations defining the code, and the other, the variable nodes, represents the elements of the codewords. A check (variable) node in the graph is referred to as degree i if it is connected to i variable (check) nodes. The fraction of edges connected to bit (check) nodes of degree i is denoted by $\lambda_i(\rho_i)$. Let L be the maximum variable degree and R the maximum check degree; we define an ensemble of LDPC codes using the generating functions $\lambda(x)$ and $\rho(x)$.

$$\lambda(x) := \sum_{i=2}^{L} \lambda_i x^{i_1},\ 0 \leq \lambda_i \leq 1$$

$$\rho(x) := \sum_{i=2}^{R} p_i x^{i-1},\ 0 \leq \rho_i \leq 1$$

We can express the code rate as a function of the coefficients of $\lambda(x)$ and $\rho(x)$:

$$Rate = 1 - \frac{\sum_{i=2}^{R} \rho_i / i}{\sum_{i=2}^{L} \lambda_i / i}$$

The functions $\lambda(x)$ and $\rho(x)$ have $L + R - 2$ non-zero coefficients. However, not all these coefficients are independent: $\lambda(x)$ and $\rho(x)$ define degree distributions and must therefore be normalized, aiming for all codes to have the same rate for the purpose of comparing their thresholds. Specifically, to ensure that $\lambda(x)$ and $\rho(x)$ define a degree distribution, we fix the coefficients corresponding to variable and check nodes of degree 2:

$$\lambda_2 = 1 - \sum_{i=3}^{L} \lambda_i, \quad \rho_2 = 1 - \sum_{i=3}^{R} \rho_i$$

The code rate can be set using a third coefficient, denoted as λL. From the expressions provided, one can derive:

$$\lambda_L = \frac{\frac{1-\beta}{R} + \sum_{i=3}^{2} \rho_i(\frac{1}{i} - \frac{1}{2}) - \beta \sum_{i=3}^{L-1} \lambda_i(\frac{1}{i} - \frac{1}{2})}{\beta(\frac{1}{L} - \frac{1}{2})}$$

where $\beta = 1 - Rate$. These constraints result in a final number of $D = L + R - 5$ parameters, each associated with one of the non-fixed coefficients of $\lambda(x)$ and $\rho(x)$. Additionally, the codes need to be stable for crossover probabilities p below their threshold, with the stability condition for the BSC channel given by [39]:

$$\lambda_2 \leq \frac{1}{2\sum_i (i-1)\rho_i \sqrt{p(1-p)}}$$

To evaluate the candidate codes, a discretized density evolution algorithm has been utilized in [36]. This algorithm calculates a threshold value for a random LDPC code with a fixed node and degree distribution, determining the error-free region limit asymptotically as the block length tends to infinity. Discretized density evolution ensures that the predicted threshold is a lower bound of

the real threshold. The results obtained with this set of constraints are presented in [36]. For all rates, the thresholds are very close to the Shannon limit. Although these thresholds are only achievable by infinite-length codes, experimental results obtained with finite-length codes were not significantly different. This aligns with the expectations, considering that the length of the codes used is quite large (106), well-suited to the typical requirements of QKD, where large blocks of data need to be processed together to minimize finite-size effects [40].

3.4 Cascade protocol

The Cascade protocol was initially introduced in [24]. In this section, we introduce the Cascade protocol, which is easily implementable.

3.4.1 Cascade Preliminaries

Binary Approach. In cases where strings A and B contain an odd number of errors, Alice and Bob can engage in an interactive binary search to pinpoint an error by exchanging fewer than $[\log n]$ bits over the public channel, as follows:

1. Alice transmits to Bob the parity of the first half of the string.
2. Bob assesses whether an odd number of errors occurred in the first half.
3. This process is iteratively applied to the determined half in step 2 until an error is located eventually.

The reconciliation protocol outlined in [23] utilizes the binary approach as the primary technique.

Confirmation Method. If Alice and Bob possess differing strings, the **Confirmation** method informs them of this fact with a probability of 3. Conversely, if their strings are identical, **Confirmation** confirms this with a probability of 1.

1. Alice and Bob independently select random subsets of corresponding bits from their strings.
2. Alice communicates to Bob the parity of her chosen subset.
3. Bob verifies that his subset exhibits the same parity.

This process can be repeated k times to provide assurance that their strings are identical. The test will fail with a probability of 2^k.

Binary Confirmation (BICONF). Combining the **Binary** and **Confirmation** approaches yields another method capable of correcting multiple errors. **BICONF** involves running **Confirmation** s times. Whenever **Confirmation** reveals a subset for which Alice's and Bob's strings have differing parities, they apply the **Binary** approach to this subset, thus correcting an error. Let $\Delta^s(l|e)$ represent the probability that **BICONF** corrects l errors given e errors. The expression is given by:

$$\Delta^s(l|e) = \begin{cases} \binom{s}{l}2^{-s} \text{ if } l \neq e \\ \sum_{j=e}^{s}\binom{j-1}{e-1}2^{-j} \text{ if } l = e \end{cases} \tag{3.1}$$

3.4.2 Cascade protocol

Cascade operates through multiple passes, and the number of passes is predetermined by Alice and Bob based on the parameter p. Let $A = A_1, \ldots, A_n$ and $B = B_1, \ldots, B_n$ (where $B_i, A_i \in (0,1)$) denote Alice's and Bob's strings, respectively.

In the first pass, Alice and Bob choose k_1 and partition their strings into blocks of k_1 bits. Block v in pass 1 is formed by the bits whose positions lie in $K_v^1 = \{l \,|\, (v-q)k_1 < l \leq vk_1\}$. Alice transmits the parities of all her blocks to Bob. Using the **BINARY** approach, Bob corrects an error in each block whose parity differs from that of Alice's corresponding block. At this point, all of Bob's blocks have an even number of errors (possibly zero). This part of the protocol is adopted from [23]. However, in that paper, leaked information about the secret string is eliminated during execution by removing one bit of each subset for which the parity is known. In our protocol, all the bits are retained. Preserving this information from pass to pass allows us to correct more errors.

In each pass $i > 1$, Alice and Bob choose k_i and a random function $f_i : [l..n] \to [1..[\frac{n}{k_1}]]$. The bits whose positions are in $K_j = \{l \,|\, f_i(l) = j\}$ form block j in pass i. Alice sends Bob:

$$a_j = \bigoplus_{l \in K_j^i} A_l$$

for each $1 \leq j \leq [\frac{n}{k_i}]$. Bob computes his b_j's in the same way and compares them with the a_j's. For each $b_j \neq a_j$, Alice and Bob execute the **BINARY** approach on the block defined by K_j^i. Bob finds $l \in K_j^i$ such that $B_l \neq A_l$ and corrects it. All the blocks K_v^u for $1 \leq u < i$ such that $l \in K_v^u$ will then have an odd number of errors. Let K be the set of these blocks. Alice and Bob can now choose the smallest blocks in K and use the **BINARY** approach to find another error. Let l' be the position of this error in strings A and B. After correcting $B_{l'}$, Bob can determine the set B formed by the blocks containing $B_{l'}$ from each pass from 1 to pass i. He can also determine the set K' of blocks with an odd number of errors by computing:

$$K' = B \triangle K$$

If $K' \neq \varnothing$, then Bob finds another pair of errors in the same way. This process is repeated until there are no more blocks with an odd number of errors, at which point pass i ends, and each block in passes 1 through i has an even number of errors (perhaps zero).

In this section, a simple analysis utilizing one of Cascade's properties demonstrates its practical utility. This analysis leads to a specific choice of block size such that the probability that a block K_v^1 has one or more errors decreases exponentially with respect to the number of passes. The

property used is that in passes following pass 1, correcting an error in K_v^1 implies that a second error from the same block K_v^1 will be corrected.

For parameters $k_1, \ldots, k_w$ chosen in a manner depending on p, we attempt to determine $\delta_i(j)$, the probability that after pass $i \geq 1, 2j$ errors remain in K_v^1. $\delta_1(j)$ is easily determined for $X \approx \text{Bin}(k_1, p)$:

$$\delta_1(j) = \text{prob}(X = 2j) + \text{prob}(X = 2j + 1)$$

Let E_i be the expected number of errors in K_v^1 after completion of pass i. For pass 1, we have:

$$E_1 = 2\sum_{j=1}^{[\frac{k_1}{2}]} j\delta_1(j) = k_1 p - \frac{(1-(1-2p)^{k_1})}{2}$$

If the functions f_i with $i > 1$ are randomly chosen from $\{f \mid f : [1, \ldots, n] \to [1, \ldots, \frac{n}{k_i}]\}$, then for $n \to \infty$, we can determine a bound on the probability γ_i of correcting at least 2 errors at pass $i > 1$ in a block K_v^1 still containing errors after completion of pass $i-1$. Since errors are corrected two by two in passes $i > 1$, we have:

$$\gamma_i \geq 1 - \left(1 - \left(1 - \frac{k_i}{n}\right)^{\frac{nE_{i-1}}{k_1}}\right)^2 \approx 1 - \left(1 - e^{-\frac{k_i E_{i-1}}{k_1}}\right)^2$$

We can bound $\delta_i(j)$ using γ_i, for $i > 1$:

$$\delta_i(j) \leq \left(\sum_{l=j+1}^{[\frac{k_1}{2}]} \delta_{i-1}(l)\right) + \delta_{i-1}(j)(1-\gamma_i)$$

Suppose that k_i is chosen such that:

$$\sum_{l=j+1}^{[\frac{k_1}{2}]} \delta_1(l) \leq \frac{1}{4}\delta_1(j) \tag{3.2}$$

and let $k_i = 2k_{i-1}$ for $i > 1$. We have:

$$\delta_i(j) \leq \frac{1}{4}\delta_{i-1}(j) + (1 - e^{-2^{i-1}E_{i-1}})^2\delta_{i-1}(j)$$

If, in addition, the choice of k_1 is such that:

$$E_1 \leq -\frac{\ln\frac{1}{2}}{2} \tag{3.3}$$

it follows that:

$$\gamma_i \geq 1 - (1 - e^{-2E_1})^2 \geq \frac{3}{4}$$

When $k_i = 2k_{i-1}, i > 1$ and k_1 satisfies 2 and 3, we have $\delta_i(j) \leq \frac{\delta_{i-1}(j)}{2} \leq \frac{\delta_i(j)}{2^{i-1}}$ since $E_i \leq \frac{E_{i-1}}{2}$.

We can bound the amount of information $I(w)$ per block of length k_i (per block K_v^1) leaked after w passes, with parameters k_i set as above (where w must not depend on n for the argument to apply), as follows:

$$I(w) \leq 2 + \frac{1-(1-2p)^{k_1}}{2}[\log k_1] + 2\sum_{l=2}^{w}\sum_{j=1}^{[\frac{k_1}{2}]} \frac{j\delta_1(j)}{2^{l-1}}[\log k_1]$$

Table 4.1 provides the values of k_1 (the largest one satisfying 2 and 3) for $p \in \{0.15, 0.10, 0.05, 0.01\}$, and the values of $I(4)$ are computed. Additionally, the average amount of leaked information $\hat{I}(4)$

for 10 empirical tests (with $n = 10{,}000$) under the same conditions is reported. For each of these tests, all errors were corrected after pass 4.

p	k_1	$\hat{I}(4)$	$kh(p)$	$I(4)$
0.01	73	6.47	5.89	6.81
0.05	14	4.60	4.01	4.64
0.10	7	3.81	3.28	3.99
0.15	5	3.80	3.05	4.12

Table 3.1. Original table of k values (source: [24]).

3.4.3 Cascade Implementations

Let's first analyze the principal steps of the most common cascade implementations.

The Cascade protocol is an iterative protocol that cycles through four steps, as mentioned before the number of iteration (i.e. cycles) to be performed in order to achieve a good error correction is variable depending on the implementation, this matter will be discussed later in this section.

Step1: Key shuffle

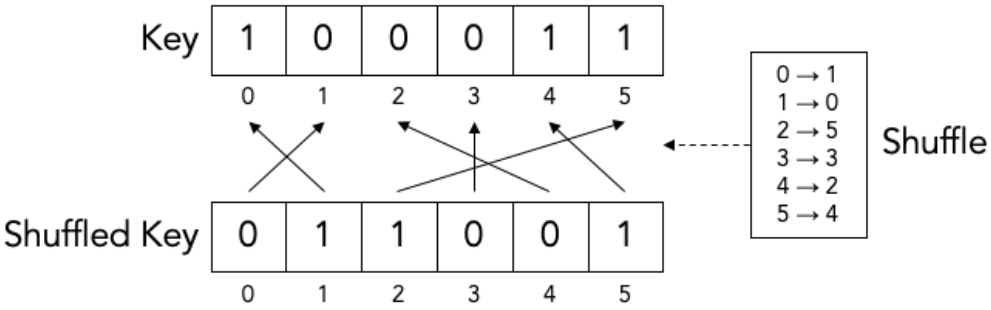


Figure 3.2. The key received by Bob contains some random errors (source: [22]).

The first step of each cycle is to shuffle the key in order to redistribute the errors in a random way (note that after the transmission some error can be packed together due to momentary noise). Since the aim is not to obscure the real order of the bits, this random shuffle can be publicly disclosed and sent to the other part.

Step2: Split the key into blocks

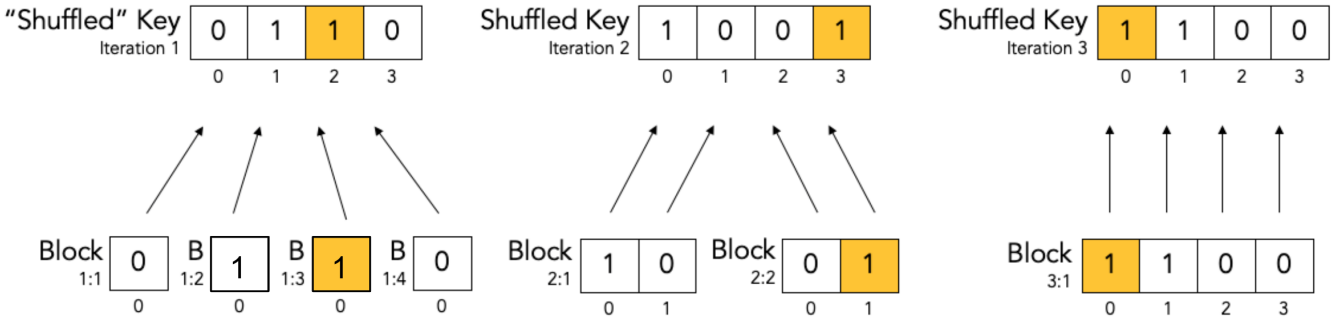


Figure 3.3. During every iteration the key is shuffled and then divided into blocks (source: [22]).

In order to binary search the errors in the key, it has to be divided into blocks. The size of the blocks in most implementation is different in every iteration, in table 4.1 it is shown that in the original design of the protocol the size of the blocks was $K = 0.73/p$ where p is the error probability i.e. the Quantum Bit Error Rate of the channel, this was considered for a key lenght of $n = 10'000$. In the original design all the subsequent block sizes where calculated as $k_i = 2k_{i-1}$.

Step3: Error search

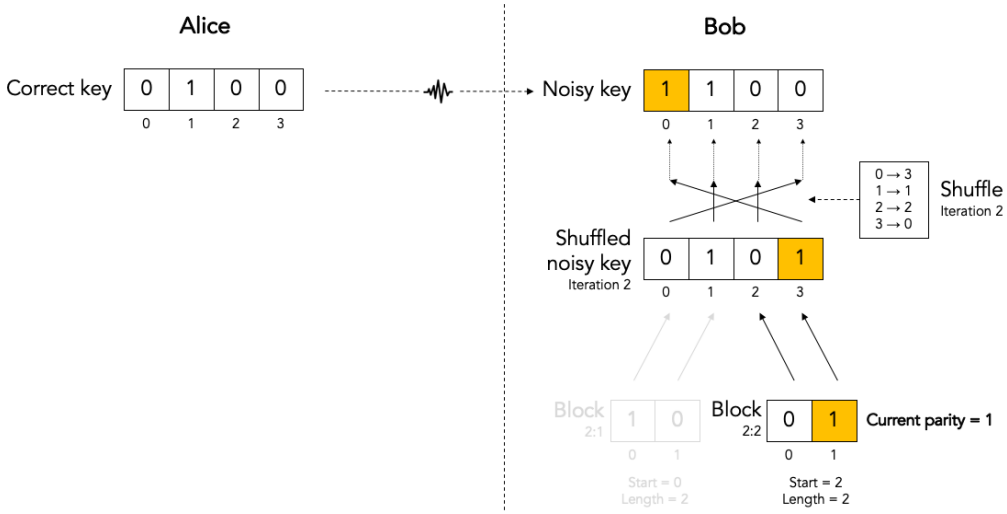


Figure 3.4. After shuffling the key, bobs computes the parity of one of the blocks (source: [22]).

Bob computes the parity bit of one of the key blocks and send it to Alice. Note that in this protocol, the parity bit is computed on a even number of bits.

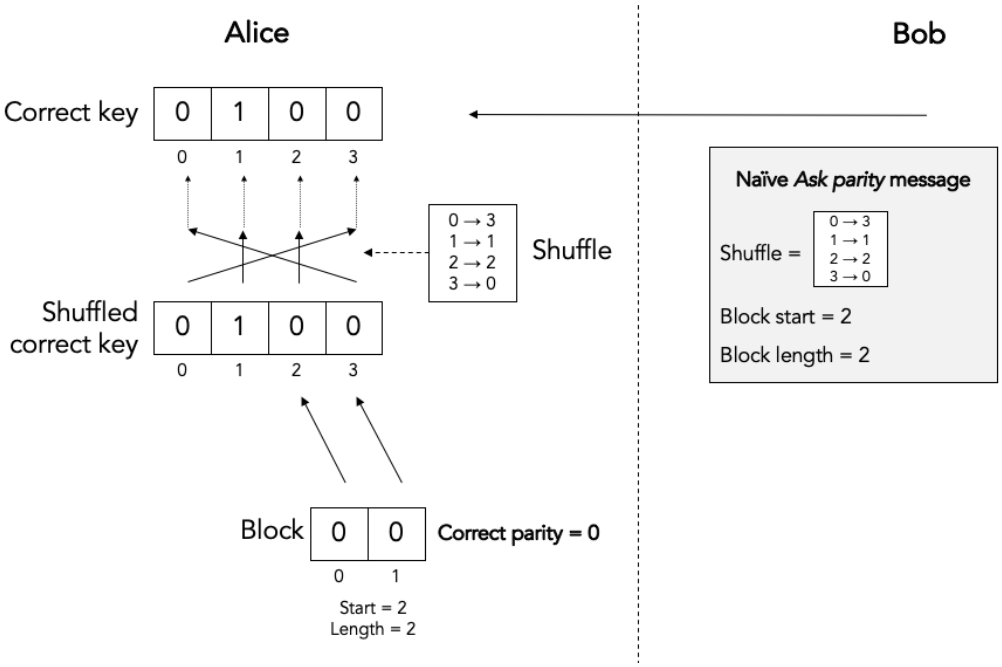


Figure 3.5. Bob then asks for the same parity bit by alice, in the figure is presented an example of a naive message (source: [22]).

Alice will use the information provided by Bob, the shuffle, the number of the first bit of the block and the block length, to calculate the correct parity bit of the same block and send it to Bob.

Step4: Error correction

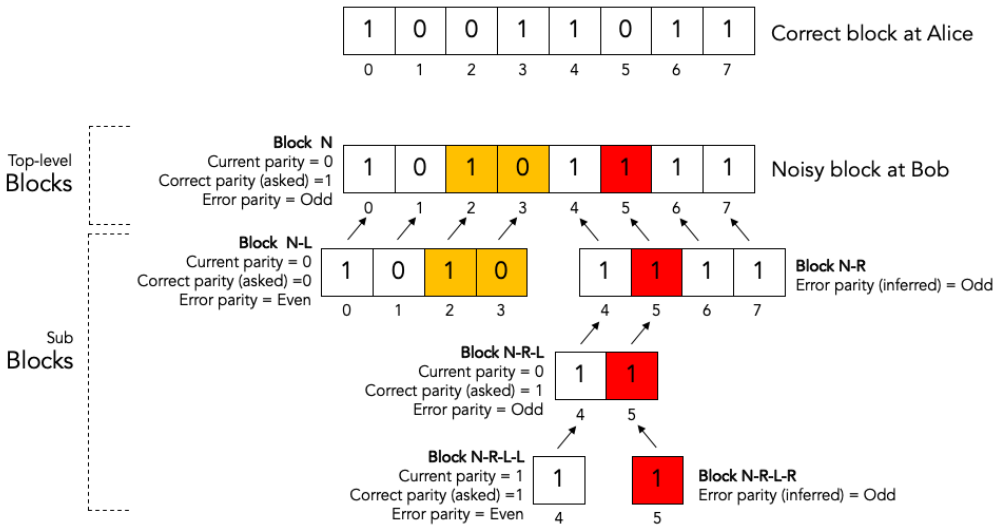


Figure 3.6. Bob corrects one error using **BINARY**(source: [22]).

If the parity bit does not match, bob initiate a binary search of the error inside the current block. The algorithm used is the **BINARY** described in the previous section.

A the end the cycle is repeated from the first step, in the original design the number of total

iterations is 4, after 4 iteration the probability of having errors inside the key was observed to be significantly low.

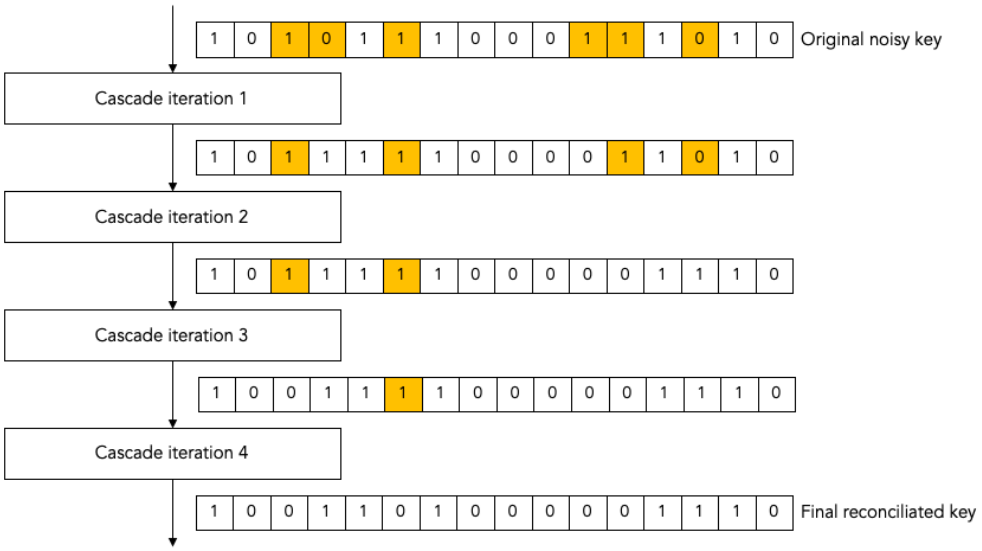


Figure 3.7. Cascade iterations scheme(source: [22]).

Different implementations

The cascade protocol has many parameters that can be changed to achieve different implementations. In this section we will see some of those parameters and their impact on the performance.

BICONF In some Cascade implementations after two steps, the iterative **BICONF** algorithm is initiated. The algorithm operates as follows: in each iteration, both parties agree upon a random subset of bits from their respective frames. They then compute and exchange the parity value of this subset. If their parities differ, two dichotomic searches are conducted, one for the chosen subset and another for the complementary subset (i.e., the subset of bits that were not selected). The algorithm selects new random subsets of bits in each iteration and halts when it has either completed a fixed number of iterations [24] (e.g., $s = 10$) or after s successive iterations without discovering new errors.

It's important to note that the exact process for selecting the random subset of bits is not specified. In our approach, we opted to choose it by performing independent Bernoulli processes with a success probability of one half for each bit of the frame. This effectively divides the frame into two subsets: a selected subset and its complement with respect to the frame, both of similar size.

In a related work [31], it's demonstrated that this modified version enhances the efficiency of Cascade. However, extensive simulations have shown that the frame error rate is notably higher in this protocol compared to the original Cascade. Thus, while the modification improves efficiency, it does so at the expense of a higher frame error rate, a characteristic often observed in one-way reconciliation with block codes. The results outlined in [31] also emphasize that a single pass of Cascade with a block size equal to half the frame length (i.e., $k_i = \lceil n/2 \rceil$) functions effectively as one iteration of **BICONF**. This presents an advantage of potentially correcting additional errors from previous passes.

Singleton blocks In an unpublished draft, the author proposes protocol optimization through enhanced random shuffling between passes and the elimination of singleton blocks in subsequent passes. Here, "singleton" denotes a sub-block with a size of one, where the value is already known either through prior exchange or deducible from other sub-blocks whose values are known.

Sub-block Reuse in [21] an optimized version of Cascade was presented, suggesting to reuse some sub-blocks created during the various iteration, to avoid asking Alice the same parities multiple times.

Indeed, during the Cascade process, there's potential to capture and utilize additional data. Currently, only the blocks initially divided at the start of each pass are documented. However, in the BINARY process, numerous smaller sub-blocks are created. If we preserve a record of all these smaller blocks, the traceback phase can search within smaller blocks. This approach reduces the exchanged information and enhances the efficiency of the protocol.

Protocol	Block sizes (approx.) k_1	k_2	k_i	`Cascade` passes	`BICONF`	Block reuse	Shuffling	Singl. blocks
orig. Ref. [6]	$0.73/Q$	$2k_1$	$2k_{i-1}$	4	no	no	random	no
mod. (1) Ref. [10]	$0.92/Q$	$3k_1$	–	2	yes	no	random	no
opt. (2) Ref. [19]	$0.8/Q$	$5k_1$	$n/2$	10	no	no[a]	random	no
opt. (3)	$1/Q$	$2k_1$	$n/2$	16	no	no	random	no
opt. (4)	$1/Q$	$2k_1$	$n/2$	16	no	yes	random	no
opt. (5)	$1/Q$	$2k_1$	$n/2$	16	no	yes	determ.	no
opt. (6)	$1/Q$	$2k_1$	$n/2$	16	no	yes	random	yes
opt. (7)	$2^{\lceil \log_2 1/Q \rceil}$	$4k_1$	$n/2$	14	no	yes	random	no
opt. (8)	$2^{\lceil \alpha \rceil}$	$2^{\lceil (\alpha+12)/2 \rceil}$	$n/2^b$	14	no	yes	random	no

Figure 3.8. Cascade implementation scheme(source: [31]).

In [31] the different cascade implementations were tested and compared, in the figures below the results are shown to give an idea on how the different parameters impact on the efficiency

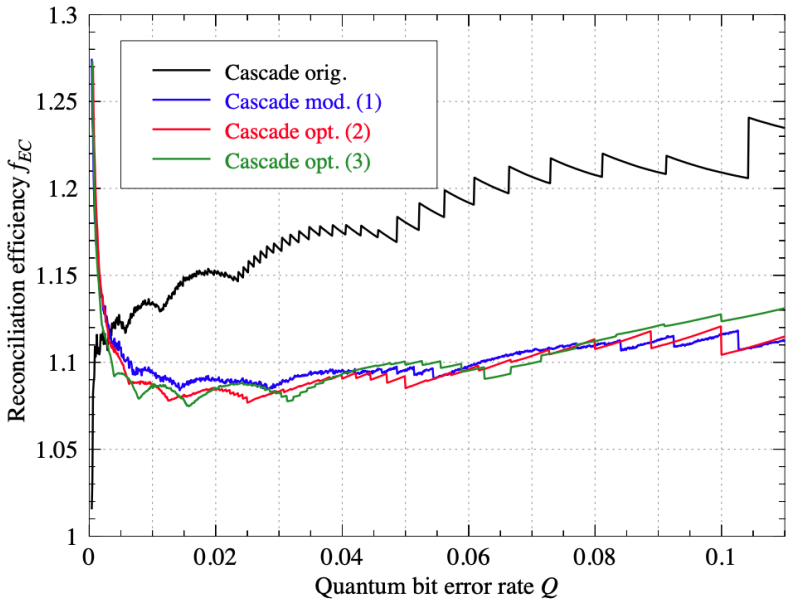


Figure 3.9. Average reconciliation efficiency (f_{EC}) for various versions of Cascade is compared: the original Cascade (in black), a modified protocol (in blue), a version with optimized parameters according to [21] (in red), and the version proposed here with 16 passes (in green) (source: [31]).

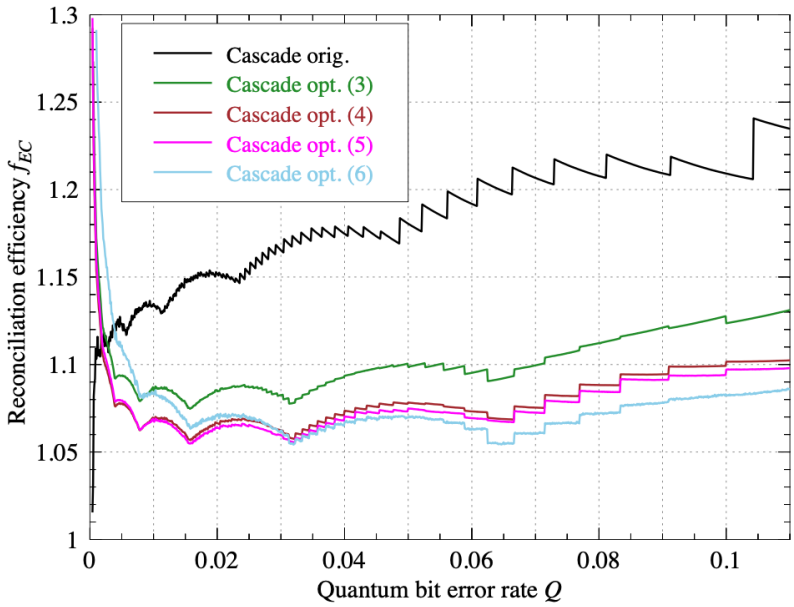


Figure 3.10. The average reconciliation efficiency (f_{EC}) for various versions of Cascade is compared: (1) Original Cascade (black) (3) Version using 16 passes, as proposed and presented in the previous figures (green) (4) Version leveraging block reuse as suggested by [21] (brown) (5) Version replacing random shuffling between passes (magenta) (6) Version discarding singleton blocks after each pass (sky blue) (Source: [31])

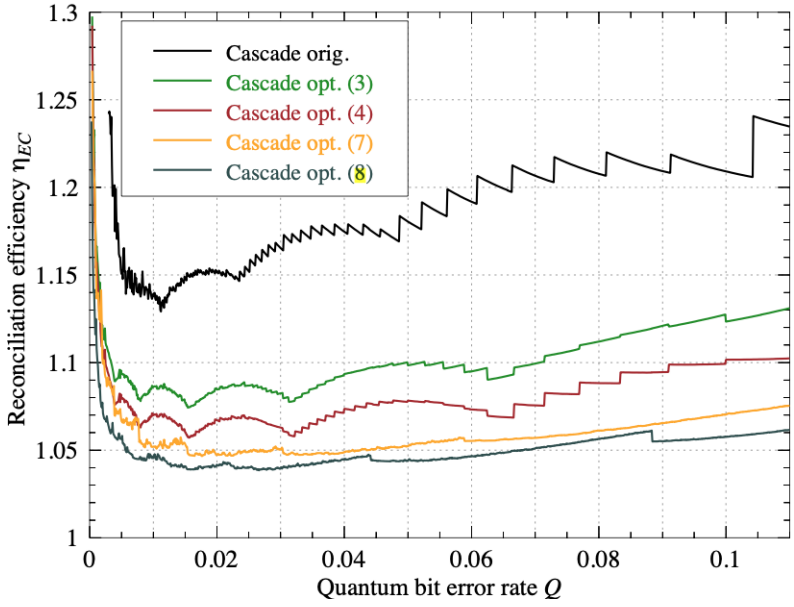


Figure 3.11. The average reconciliation efficiency (η_{EC}) for various versions of Cascade is compared: (1) Original Cascade (black) (3) Version using 16 passes as proposed above and presented in the previous figures (green) (4) Version leveraging block reuse in addition to (2) (brown) (7) Version optimizing the first and second block sizes and using 14 passes (orange) (8) Version optimizing the third block size and using a power of two value for the frame length $n = 2^{14}$ (dark gray) (Source: [31])

Chapter 4

Analysis of Privacy Amplification protocols

4.1 QKD Privacy amplification

Quantum key distribution (QKD), based on the uncertainty principle and the No-Cloning theorem, theoretically has higher security than the existing information security schemes [41]. However, the generated key itself has no substantive information. Only when it is encrypted by the encryption algorithm as a key can the information required by both sides of the communication be transmitted [42, 43, 44, 45]. The encrypted key needs to be transmitted in the public channel, which inevitably leads to the risk of information disclosure. In order to delete the leaked information from the negotiation key containing the leaked information, Bennet et al. proposed an important privacy amplification step in the post-processing of quantum communication[9, 19], which realizes the unconditional security of the quantum key distribution system by compressing the negotiation key into an absolutely secure final key [46, 47, 48].

A common PA is to compress a string of keys through a universal hash function, and then eliminate the information leaked to attacker Eve. In this way, the security key can be obtained. The hash function is usually selected as the Toeplitz matrix, whose element is 0 or 1 [49]. Some researchers use a variety of acceleration software methods provided by fast Fourier transform (FFT) [50] to realize the privacy amplification algorithm through CPU and GPU software [51]. Its experimental efficiency is relatively good and can achieve a considerable processing rate. In [50], the researchers propose a FFT PA scheme on commercial CPU platform. The long input weak secure key is divided into many blocks, then PA procedures are parallel implemented for all sub-key blocks, and afterwards the outcomes are merged as the final secure key, but FFT also needs to consume a lot of computing resources. Moreover, for the practical quantum key distribution system, these methods of using CPU software to realize the PA algorithm have hidden dangers in security. There may be various unknown backdoors and vulnerabilities in this system, which greatly affect the work of quantum key distribution system. Therefore, researchers propose to use a field programmable gate array (FPGA) platform to implement the privacy amplification algorithm. The algorithm implemented by FPGA is a pure hardware logic circuit with low security risks. In [**?**], Lu et al. proposed a PA algorithm implemented on FPGA platform. By constructing the required Toeplitz matrix on FPGA and using the characteristics of FPGA to calculate the Toeplitz matrix in parallel, they succeed in improving the running speed of the algorithm and the maximum safe coding rate of the system. In addition, the algorithm can also achieve any number of input key bits in a certain length, which is helpful for the implementation of future PA algorithms [52]. In [53], Toepliz matrix is divided into several sub blocks, and FPGA is used to process the sub blocks in parallel to improve the operation speed. However, it only considers the reconstruction of the Toepliz matrix, and does not involve the adequate processing of the negotiated key with the Toepliz matrix. In view of the high requirements of hardware resources and low computing speed of Toepliz matrix, researchers put forward some effective schemes to improve it. In [54], they propose a privacy amplification algorithm based on LFSR to save storage

space and speed up operation process. For the storage of elements in the Toepliz matrix, only one register is needed, which greatly saves hardware storage resources. In [55], Bai et al. proposed a PA algorithm based on Toeplitz matrix, which uses LFSR to save storage space and speed up the privacy amplification process. The continuous state transformation of LFSR is constructed. The results of each LFSR state are accumulated at the same time of LFSR state transition. Repeat the above steps through block iteration to obtain the final key. Because the operations of different accumulators are independent, the calculation of the final key is parallel, and the speed of the algorithm can be improved. However, due to the characteristics of its sequential transformation to produce the whole Toepliz matrix, the rate of generating the final key is still inevitably affected.

In [20], they propose a PA algorithm based on Cellular Automata (CA) and block structure. CA is used to generate a pseudorandom sequence with good random characteristics. The sequence performs a bit operation with the negotiation key and accumulates in blocks, so as to realize the function of compressing the longer key into the final key. Unlike the algorithm of dynamically generating Toeplitz matrix using LFSR, CA does not need to generate random sequences bit by bit like LFSR. Due to the characteristics of CA, it can generate many new random sequences in parallel, which improves the operation speed and can generate keys of any length. The National Institute of Standards and Technology (NIST) randomness test [56] and avalanche test show that the final key generated by the algorithm also has good randomness performance and a good avalanche effect [57, 58].

In this chapter the main Privacy Amplifications algorithms will be analyzed.

4.1.1 PA metrics

The privacy amplification process involves compressing the negotiation key, acquired during the quantum key distribution process, into a shorter final key. This aims to remove potential leaked information in the conventional channel, ensuring unconditional security [59]. In this step, eavesdropper Eve finds it extremely difficult to gather any meaningful information about the key, ensuring that the secure key used by Alice and Bob remains highly secure.

Viewed through the lens of information theory, the PA process can be seen as a technique for extracting highly confidential shared information, the security key, from a larger pool of partially secure shared information at risk of exposure. Suppose Alice and Bob share a random variable W, such as a random bit string of length n. Eve, influenced by various factors, can obtain a maximum of t bits of information about W, where $t \leq n$. This is represented by $H(W|V) \geq n-t$. However, Alice and Bob usually don't have detailed knowledge about the distribution of these random variables. They aim to publicly choose a compression function $g : Sn[0,1] \rightarrow Sr[0,1]$ where $(n > r)$, satisfying certain constraints, to minimize Eve's information about W based on her knowledge of the compression function g. Through this process, the key $k = g(V)$ is generated, and due to its nearly uniform distribution, Eve can't glean information about W from k [60].

$$I(k : g, V) \approx 0$$

Therefore, the key obtained after the PA algorithm can be safely used as the encryption key. One of the main concerns about Privacy Amplification is the running time of some algorithms and their computational cost, this will be the main metric in comparing them.

4.2 FFT

Privacy amplification was initially introduced within quantum key distribution by [23]. It addresses a scenario where a channel has perfect authenticity but lacks privacy (a public classical channel) and aims to rectify the deficiencies of a quantum channel, which offers imperfect privacy but no authenticity. In [61] an implementation of Privacy amplification using Fast Fourier Transform was presented as follows: Alice and Bob start by transmitting quantum signals over a noisy and lossy quantum channel (fiber or free space). Following basis/key sifting and error correction

procedures, they share a correlated yet weak secure key denoted as W through a public channel. The min-entropy of this shared weak secure key W is n. Let E be a random variable summarizing Eve's knowledge about W. Here, $H(W|E) \leq t$, where $t < n$. Privacy amplification involves Alice and Bob discussing an extractor function $G : \{0,1\}^n \to \{0,1\}^r$ publicly. This function reduces Eve's knowledge of the final secure key K_f from t to at most ϵ. In contemporary applications, most practical extractors are implemented using universal hash functions, particularly the (modified) Toeplitz matrix [62].

The Toeplitz matrix $T(A)$ can be defined as follows:

$$G(A) := (I_r|T(A)) = \begin{bmatrix} 1 & & & a_{r-1} & a_r & ... & a_{n_2} \\ & 1 & & a_{r-2} & a_{r_1} & ... & a_{n-3} \\ & & ... & ... & ... & ... & ... \\ & & 1 & a_0 & a_1 & ... & a_{n-r-1} \end{bmatrix}$$

where A is a random seed represented as $A = (a_0, a_1, ..., a_{n-1}) \in \{0,1\}^{n-1}$, and $T(A)$ is a $r \times (n-r)$ Toeplitz matrix. Moreover, $W_I = (w_0, w_1, ..., w_{r-1})$ and $W_{TA} = (w_r, w_{r+1}, ..., w_{n-1})$. The final secure key can be calculated as:

$$K_f = G(A)W = I_r \times (w_0, w_1, ..., w_{r-1}) \oplus T(A) \times (w_r, w_{r+1}, ..., w_{n-1}) = W_I \oplus T(A)W_{TA}$$

To efficiently calculate $T(A)W_{TA}$ using the Fast Fourier Transform (FFT), $T(A)$ is extended to a special circulant Toeplitz matrix of scale $(n-1) \times (n-1)$, and W_{TA} is extended to a vector of length $n-1$ by padding zeros. The optimized multiplication of a circulant matrix and a vector is represented as:

$$H \cdot X = F^{-1}[F(h) * F(X)]$$

where $*$ denotes the Hadamard product operator, F represents the Fourier transform operator, F^{-1} is the inverse Fourier transform operator, X is a vector, and H is a circulant Toeplitz matrix with the first row h. The computational complexity of the optimized privacy amplification algorithm using this approach is $O(n \log n)$ [63, 64].

In theory, quantum key distribution can generate secure keys for communicating parties, even if the quantum channel is under Eve's control. However, imperfect implementation and active attacks may leak some information about W to Eve. Alice and Bob can accurately quantify the bound of leaked information with an infinite post-processing block size. For entanglement-based quantum key distribution, the secure key rate can be calculated as follows[65]:

$$R \geq qQ_\mu \nu_s [1 - H_2(e_p^U) - f(e_b)H_2(e_b)]$$

where q is the basis sifting factor, Q_μ is the gain of detected entangled photon pairs, ν_s is the repetition rate of the entangled source, e_b is the measured quantum bit error rate, e_p^U is the estimated upper-bound of the phase error rate, $f(x)$ is the error correction efficiency, and $H_2(x)$ is the binary Shannon entropy.

In practical scenarios, e_p^U cannot be measured directly and accurately estimated due to statistical fluctuations with finite post-processing block sizes. A simulation of the required throughput of the privacy amplification algorithm was conducted in a 10 GHz entanglement-based quantum key distribution with the parameters shown in Table 1. The entangled photon source was placed in the middle of the communicating parties, considering the finite-size effect for the final secure key K_f with a post-processing block size ranging from the order of 10^4 to infinite. Additionally, a failure probability $\epsilon^{ph} = 10^{-10}$ was used for estimating e_p^U. The analyzed results, indicate that the post-processing block size should be at least on the order of 10^8 to achieve a secure key rate close to the asymptotic limit. Directly implementing privacy amplification algorithms with ultra-large-scale inputs will constrain the performance of full quantum key distribution systems. Meanwhile, the required throughput of the privacy amplification algorithm is approximately 40 Mbps without any channel loss.

4.2.1 HiLS Scheme

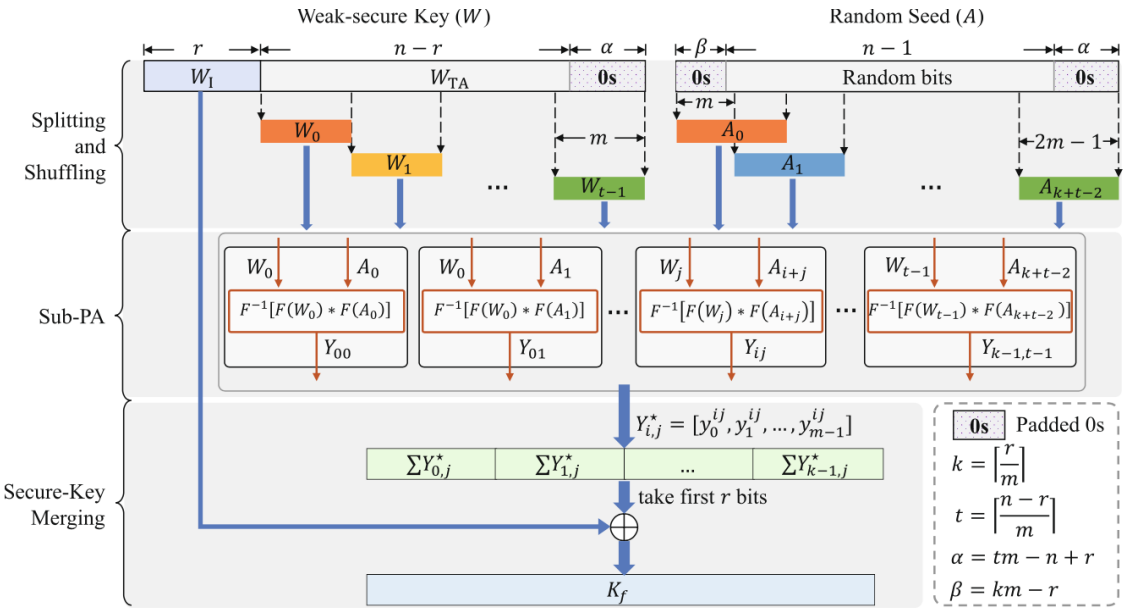


Figure 4.1. HiLS Algorithm scheme(source: [61]).

The diagram illustrating the high-speed and large-scale (HiLS) privacy amplification scheme for Quantum Key Distribution (QKD) is depicted in Figure 4.1. The weak secure key W, having a length of n, is obtained after undergoing basis/key sifting and error correction procedures based on the measured raw key string at Alice's (or Bob's) end. Subsequently, Alice and Bob estimate the final secure key length r using a meticulous statistical fluctuation analysis. Following this, Alice and Bob engage in a public discussion regarding a random seed with a length of $n-1$ bits to construct the universal hash function. The HiLS Privacy Amplification (PA) scheme primarily comprises three steps: splitting and shuffling, sub-PA, and secure-key merging.

Step 1: Splitting and Shuffling

In this step, W is divided into several sub-vectors, and the Toeplitz matrix $T(A)$ is divided into sub-matrices. Assuming the scale of the sub-matrix is $m \times m$ where $m \leq r$, the Toeplitz matrix $T(A)$ can be divided into t blocks by rows and k blocks by columns. This results in a total of kt sub-matrices, where $t = \lfloor n - r \rfloor$, and $rmk = \lfloor m \rfloor$. Initially, they construct a vector A by padding $km - r(tm - n + r)$ zeros to the head (tail) of the exchanged random seed with a length of $n-1$ bits. Afterward, they shuffle A into $k + t - 1$ sub-vectors, denoted as $A_i := [a_{im}, a_{im} + 1, ..., a_{(2+i)m-1}], 0 \leq i < k + t - 1$. Consequently, the divided sub-matrix can be constructed using $H_{i,j} = T(A_{i+j}), i \in [0, k)$ and $j \in [0, t)$. This yields:

$$T(A) = \begin{bmatrix} H_{k-1,0} & H_{k-1,1} & ... & H_{k-1,t-1} \\ H_{k-2,0} & H_{k-2,1} & ... & H_{k-2,t-1} \\ \vdots & \vdots & \vdots & \vdots \\ H_{00} & H_{01} & ... & H_{0,t-1} \end{bmatrix} = \begin{bmatrix} T(A_{k-1}) & T(A_k) & ... & T(A_{k+t-2}) \\ T(A_{k-2}) & T(A_{k-1}) & ... & T(A_{k+t-3}) \\ \vdots & \vdots & \vdots & \vdots \\ T(A_0) & T(A_1) & ... & T(A_{t-1}) \end{bmatrix}$$

where $H_{i,j} = H_{i+1,j+1}$. For W, they initially pad $tm - n + r$ zeros to the tail and take the first r bits, and the remaining bits are used to construct the sub-vector W_I and W_{TA}. They then divide W_{TA} into t sub-vectors, defined as $W_i := [w_{im+r}, w_{im+r+1}, ..., w_{(i+1)m+r-1}]$, where $0 \leq i < t$.

Step 2: Sub-PA

In this step, the efficient implementation using Fast Fourier Transform (FFT) is performed on the multiplication of $Y_{i,j}$ to the sub-vector W_j and sub-matrix $H_{i,j}$:

$$Y_{i_j} := F^{-1}[F(A_{i+j}) * F(W_j)]$$

where, $i \in [0,k)$ and $j \in [0,t)$.

Step 3: Secure-Key Merging

Initially, they only consider the first m bits of $Y_{i,j}$ (defined as $Y^a_{i,j}$). They then merge $Y^a_{i,j}$ into vector Z using the formula:

$$Z = (\sum_{}^{t-1} Y^*_{0,j} | \sum_{}^{t-1} Y^*_{1,j} | ... | \sum_{}^{t-1} Y^*_{k-1,j})$$

By considering the first r bits of Z (defined as Z^*), they obtain the final secure key K_f:

$$K_f = W_i \oplus Z^*$$

The detailed implementation of the HiLS PA scheme is presented in Figure 4.2. During the execution of the proposed HiLS PA scheme, they only need to perform $k + 2t - 1$ Fourier operations with a scale of $2m$, kt times Hadamard product operations with a scale of m, kt times inverse Fourier operations, and $kt + 1$ times exclusive OR (XOR) operations with a scale of m. Consequently, the computational complexity of the proposed HiLS PA scheme is $O(ktm \log m)$, simplified to approximately $O(n \log m)$.

Input: the weak secure key W with length of n, the final key length r, the random seed S with length of $n-1$ and the sub-block size m

Output: the final secure key K_f

1: $t = \lceil \frac{n-r}{m} \rceil$, $k = \lceil \frac{r}{m} \rceil$
2: $W = (W|(\mathbf{0})_{tm-n+r})$, $A = ((\mathbf{0})_{km-r}|S|(\mathbf{0})_{tm-n+r})$
3: $W_{\mathrm{I}} = (w_0, w_1, \ldots, w_{r-1})$, $W_{\mathrm{TA}} = (w_r, w_{r+1}, \ldots, w_{tm-1})$
4: $\forall i \in [0,t)$, do $W_i = [w_{im+t}, w_{im+t+1}, \ldots, w_{(i+1)m+t-1}]$ and $P_i = F(W_i)$
5: $\forall i \in [0,k+t)$, do $A_i = [a_{im}, a_{im+1}, \ldots, a_{(2+i)m-1}]$ and $Q_i = F(A_i)$
6: **while** $0 \le i < k$ and $0 \le j < t$ **do**
7: $\quad Y_{i,j} = F^{-1}[Q_{i+j} * P_j]$
8: **end while**
9: $\forall i \in [0,k)$, do $Y^\star_{i,j} = \left[y_0^{ij}, y_1^{ij}, \ldots, y_{m-1}^{ij}\right]$ and $U_i = \sum_{j=0}^{t-1} Y^\star_{i,j}$
10: $Z = (U_0|U_1|\ldots|U_{k-1})$, $Z^\star = [z_0, z_1, \ldots, z_{r-1}]$
11: $K_f = W_{\mathrm{I}} \oplus Z^\star$

Figure 4.2. HiLS Algorithm pseudo-code (source: [61]).

4.3 LFSR

In [66], they present two illustrative examples to introduce the universal class of hash functions briefly. It's important to note that regardless of whether it's CVQKD or DVQKD, every character in the reconciled key string is either a 0 or a 1. This makes it unnecessary to differentiate between privacy amplification in CVQKD and DVQKD. In essence, the process of privacy amplification remains consistent across both, even though CVQKD and DVQKD differ significantly before this step. In the privacy amplification process, the binary key string is compressed using a hash

function to eliminate information leakage to Eve. Given the primary focus on CVQKD in this work, they will primarily delve into privacy amplification within this context.

Let's consider a simple scenario: Alice randomly selects a pair of bits and calculates their exclusive-or value. It's crucial to emphasize that Alice does not disclose this exclusive-or result over the common channel. Instead, she only announces which bits she has chosen, for example, the 100th bit and the 105th bit, and then substitutes these bits with the exclusive-or result. This approach not only compresses the key sequence but also ensures consistency between the keys at both ends. Additionally, if the eavesdropper Eve has obtained partial information about the bit pair, the leaked exclusive-or information to Eve is minimized. For instance, if Eve knows only one bit of the bit pair and not the other, she cannot deduce the exclusive-or value. In another scenario, if the probability of Eve knowing both of the two bits is 60%, then the probability she guesses it correctly is $60\%^2 + 40\%^2 = 52\%$. The information leaked to Eve decreases as this process is repeated continuously. These examples demonstrate that the universal class of hash functions effectively reduce the information leaked to Eve while compressing the key sequence.

In essence, a universal class of hash functions refers to a set of functions that map larger ranges to smaller ones. A desirable characteristic of such hash functions is that when the hash values of two entities are equal, the two entities themselves are considered equal. In simple terms, since universal class of hash functions can compress the length of a set, two sets are regarded as the same if their abbreviations match. Universal class of hash functions find extensive use in achieving efficient average performance across various applications, especially in associated memory systems like compiler symbol tables or databases. One crucial application of universal class of hash functions is secure information authentication within a system, allowing the message recipient to verify the authenticity of the message, ensuring it hasn't been altered or forged by an unauthorized entity [67].

As a specific type within the universal class of hash functions, the Toeplitz matrix has found widespread adoption by many research groups [53, 51, 68, 69] new to implement privacy amplification. Considering that the input for privacy amplification is the reconciled key and the output is the final secret key, they can obtain the final secret key by multiplying the reconciled key with the Toeplitz matrix. This process transforms the reconciled key, with a length equal to the column count of the Toeplitz matrix, into the final secret key, with a length equal to the row count of the Toeplitz matrix. Generally, the column count of the Toeplitz matrix exceeds the row count, resulting in a shortened key sequence and reduced information leakage to Eve. The Toeplitz matrix is advantageous due to its simple and parallel computing features, making it easy to implement in hardware. Moreover, techniques like number theoretic transform and fast Fourier transform (FFT) offer various approaches to accelerate its software implementation [51, 70]

4.3.1 Finite-size Effect on Privacy Amplification

To ensure the security of the final secret key, they must consider the finite-size effect. Let's briefly analyze this effect and its impact on the final secret key rate. Assuming x and y represent the classical variables of Alice and Bob after measurement, and E represents the quantum states of the eavesdropper Eve. The final secret key rate in CVQKD can be expressed using the formula:

$$k = \beta I(x:y) - S(y:E) - \Delta(n) \tag{4.1}$$

Here, β represents the reconciliation efficiency, $I(x:y)$ is the mutual entropy between the data of Alice and Bob, $S(y:E)$ is the von Neumann entropy of Bob and Eve, n is the length of the reconciled key, and $\Delta(n)$ represents the influence of the finite-size effect on the security of privacy amplification. The value of $\Delta(n)$ can be calculated using the following formula:

$$\Delta(n) = (2dimH_x + 3)\sqrt{\frac{\log_2^{2/\hat{\epsilon}}}{n}} + \frac{2}{n}\log_2^{(\frac{1}{\epsilon_{PA}})} \tag{4.2}$$

In this context, they introduce the use of Hilbert space H_x corresponding to variable x, where n represents the length of the reconciled key, $\hat{\epsilon}$ is a smoothing parameter, and ϵ_{PA} denotes

the failure probability of the privacy amplification process. It's important to note that both ϵ and ϵ_{PA} are intermediary parameters that can be optimized to meet security requirements. As shown in equation (4.2), the length of the reconciled key (n) significantly influences $\Delta(n)$ under fixed conditions for other parameters. Furthermore, $\Delta(n)$ decreases with an increase in n. Similarly, as per equation (4.1), the secret key rate K increases with a decrease in $\Delta(n)$. Combining these points, when the reconciled key length (n) is sufficiently large, the secret key rate is minimally affected. Conversely, a short reconciled key greatly reduces the secret key rate. Privacy amplification effectively eliminates almost all the information leaked to Eve. Therefore, reducing the finite-size effect ($\Delta(n)$) is crucial to enhance the actual secret key rate. However, as the length of the reconciled key increases, the Toeplitz matrix also expands. This expansion poses challenges regarding storage for Toeplitz matrix elements and the speed of privacy amplification.

4.3.2 Costruction of Toepliz Matrix Based on LFSR

Let $A = GF(2)^l, B = GF(2)^k$. Let M be a $k \times l$ matrix and x be a vector. The product of M and x can be expressed as $h_M(x) = Mx$. Then$R = \{h_M : M \in GF(2)^{k\times l}\}$ is universal class of hash functions. Particularly, they select Toeplitz matrix as M. As one of the universal hash functions, Toeplitz matrix is easy to be constructed. Since the elements belonging to the same diagonal line are equal, Toeplitz matrix is also called diagonal-constant matrix. More concretely, in a Toeplitz matrix $M_{k\times l}(l > k)$,if for $\forall i, j, \delta \in N$ and $1 \le i, i+\delta \le k, 1 \le j, j+\delta \le l$, it has $M_{i,j} = M_{i+\delta,j+\delta}$.Considering the above properties, they can obtain the Toeplitz matrix which they can recall is as follows:

$$\begin{bmatrix} 1 & & & a_{r-1} & a_r & \dots & a_{n_2} \\ & 1 & & a_{r-2} & a_{r_1} & \dots & a_{n-3} \\ & & \dots & \dots & \dots & \dots & \dots \\ & & 1 & a_0 & a_1 & \dots & a_{n-r-1} \end{bmatrix}$$

It can be seen from the above matrix that the Toeplitz matrix can be uniquely determined by initializing the first line and the first column of a matrix, so they only need to store $k + l - 1$ elements. In CVQKD, they should guarantee the length of the key sequence is long enough to reduce the finite-size effect, so they still need prodigious computer resources to store elements of the Teoplitz matrix.

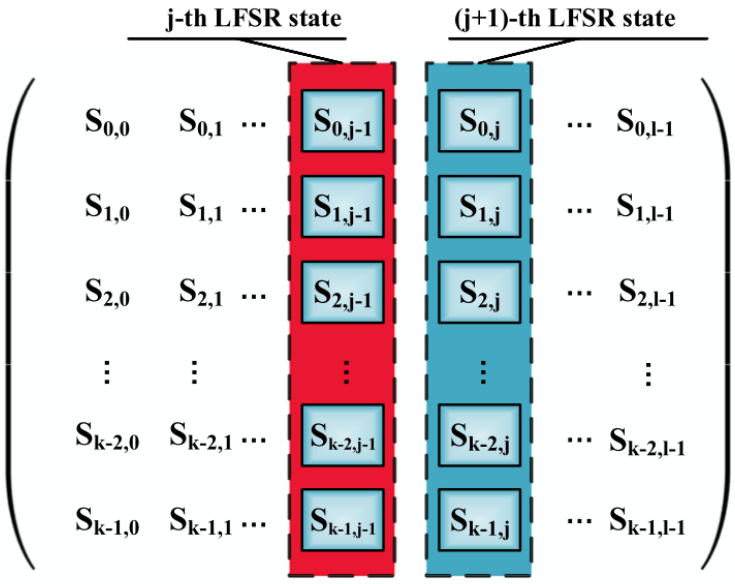


Figure 4.3. The relationship between LFSR states and the columns of the Toeplitz matrix is such that consecutive LFSR states correspond to consecutive columns of the Toeplitz matrix. (source: [66]).

The Linear Feedback Shift Register (LFSR) is a technique for creating Toeplitz matrices. Historically, this method found use in authentication due to its notably lower implementation complexity, key size, and randomness [71]. The number of bits in the register corresponds to the row count of the Toeplitz matrix, forming an LFSR state with its current value. Notably, within the Toeplitz matrix $M_{k\times l}$, the successive columns represent consecutive LFSR states of length k, resulting in a total of l LFSR states. Since each column (except the first) in the Toeplitz matrix can be derived by shifting the previous column downwards and adding a new element at the top, employing LFSR presents an approach to construct the Toeplitz matrix. The subsequent LFSR state is obtained by shifting one unit down and updating the value of the first bit in the register. Figure 4.3 elucidates the relationship between Toeplitz matrix and LFSR states. Unlike the usual $k \times l$ matrix, a Toeplitz matrix defined by only $k+l-1$ elements remarkably saves storage space. Nevertheless, the cost remains substantial when the input length significantly exceeds the output. The Toeplitz matrix based on LFSR is determined by continuous LFSR states with length k and requires only a k-bit register. The LFSR-based method involves the following steps:

1. Initialization of the first column of the matrix (1st LFSR state).
2. Shifting down by one unit of the first LFSR state.
3. Addition of an element to the top position of the second LFSR state.
4. Repetition of the above process until the last LFSR state (last column of Toeplitz matrix) is determined.

The initialization involves randomly selecting binary strings of length k. Indeed, an irreducible polynomial can be set to control the feedback [72]. For the selection of irreducible polynomials, Ref. [71] offers a practical scheme to find a set of irreducible polynomials of degree n. Random selection of one of them suffices.

Theorem 2: For $Z_p = \langle\{0, 1, ..., p-1\}, +, \cdot\rangle$, with operations limited to addition and multiplication mod p, let $g(x) = x^n + a_{n-1}x^{n-1} + ... + a_0$. Consider $e_1, ..., e_k$ as all the prime divisors of degree n and $m_i = n/e_i$ for $1 \leq i \leq k$. The polynomial $g(x)$ is irreducible if and only if:

$$g(x)|(x^{p^n} - x), (g(x), x^{p^{m_j}} - x) = 1, 1 \leq i \leq k$$

Where (A, B) denotes the greatest common divisor of A and B. By applying this scheme and leveraging the robust computational capacity of computers, it becomes convenient to identify a set of irreducible polynomials of degree n.

Assuming $p(x)$ is an irreducible polynomial with length k and coefficients $p_{k-1}, p_{k-2}, ..., p_0$ over $GF(2)$ domain, and the initialization state of LFSR is $s_{0,0}, s_{1,0}, ..., s_{k-1,0}$. Following the steps mentioned earlier, the top element of the $(j+1)$-th LFSR state is given by:

$$s_{0,j} = \bigoplus_{i=0}^{k-1} s_{i,j-1} \cdot p$$

The preceding description elucidates the transition between consecutive LFSR states and the construction process of the Toeplitz matrix using the LFSR method. Figure 4.4 details the transition. By utilizing the transition of LFSR states, all columns of the Toeplitz matrix can be obtained, facilitating matrix determination. Consequently, only a k-bit register is necessary instead of $k+l-1$ storage units. Here, they present a simple example to illustrate the process. Let's assume the length of the reconciled key is 6 and the final secret key is 4. Choosing the irreducible polynomial $p(x)$ as $x^3 + x + 1$ and its corresponding vector as $x_p = (1, 1, 0, 1)^T$. They randomly select binary strings $1, 0, 0, 1$ of length 4 as the first LFSR state. Subsequently, the elements of the first column (from top to bottom) of the Toeplitz matrix H_{4x6} are 1,0,0,1 and the corresponding vector of the first LFSR state is $u_0 = (1, 0, 0, 1)$.

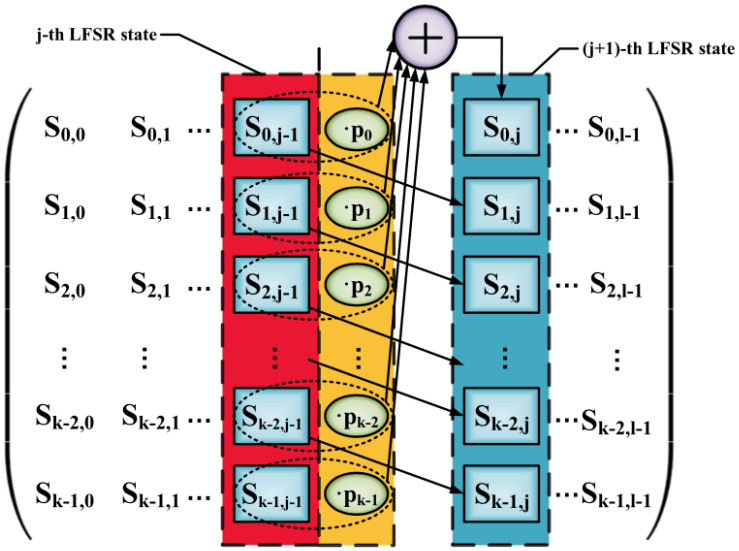


Figure 4.4. The transition between two adjacent LFSR states (source: [66]).

By shifting down, they derive that the second to fourth elements of the second LFSR state are 1,0,0. Furthermore, the first element of the second LFSR state is $u_0^T \cdot x_p = 0$. Consequently, the second LFSR state is 0,1,0,0, and they designate it as the second column of the Toeplitz matrix. Similarly, the second to fourth elements of the third LFSR state can be obtained by shifting down the second LFSR state, resulting in 0,1,0. The vector corresponding to the second LFSR state is $u_1 = (0, 1, 0, 0)^T$, and the product value $u_1^T \cdot x_p = 1$ serves as the first element of the third LFSR state. Thus, the third LFSR state is 1,0,1,0, representing the third column of the Toeplitz matrix. This process is repeated to obtain the remaining 3 LFSR states, ultimately determining the complete Toeplitz matrix, as presented below:

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

In a typical Toeplitz matrix, security hinges on the randomness of the elements in the first column and the first row. Conversely, the method initializes solely the first column of the Toeplitz matrix. However, the randomness in this approach extends not only to the random initialization of the first column but also to the process of randomly selecting the irreducible polynomial $p(x)$ from the irreducible polynomials set of degree n. Given the comprehensiveness of the irreducible polynomials set, the Toeplitz matrices constructed using this set encompass all hash functions facilitating the establishment of the equality $h_p(M) = c$. Therefore, the randomness of the Toeplitz matrix is assured. For a more detailed security proof, refer to [72].

4.3.3 Privacy Amplification with LFSR-Based Toeplitz Matrix

Since the LFSR method enables equivalent hashing with lower costs in implementation, key size, and randomness, it was previously utilized in information authentication [72]. Similarly, as a method for constructing hash functions, LFSR can be employed in privacy amplification for CVQKD to enhance efficiency. This method it's proven advantageous, especially when the Toeplitz matrix is large, as it only necessitates one column for constructing the matrix.

In the privacy amplification algorithm, the final key is acquired through the product of the Toeplitz matrix and the reconciled key. Considering the properties of matrix multiplication and

the relationship between LFSR states and the Toeplitz matrix, the final secret key can be computed through the following linear combination:

$$\bigoplus_{j=0}^{l-1} M_j \cdot (s_{0,j}, s_{1,j}, ..., s_{k-1,j}) \tag{4.3}$$

In this context, $s_{0,j}, s_{1,j}, ..., s_{k-1,j}$ represent the $j+1$-th state of the Linear Feedback Shift Register (LFSR), $M = M_0, M_1, ..., M_{l-1}$ is the binary string of the reconciled key, and M_j is the $j+1$-th bit of this reconciled key. Regarding equation (4.3), they can implement it using a specific approach. Suppose M is the reconciled key with m bits, N is the final secret key with k bits, and H is the Toeplitz matrix in use. The number of columns of H, denoted by H_{kxm}, corresponds to m, and the number of rows corresponds to k. Each column of this Toeplitz matrix, apart from the first, is determined by shifting the previous column and adding a new element calculated from the elements of the preceding column and a chosen irreducible polynomial. When implementing the privacy amplification algorithm, they establish k accumulators to store the final secret key, initializing them to 0. They then set up a linear feedback shift register to generate a linear shift sequence. Based on the input bit M_j. They decide whether the elements of the $j+1$-th column of the matrix should be placed into the accumulators. If $M_j = 1$, They include the elements of the $j+1$-th column in the accumulators. If $M_j = 0$, no action is taken. Essentially, the message bit of the reconciled key serves as an enable signal, directing the operation of the accumulators. The final secret key can be computed while transitioning through LFSR states. After m time periods, the final result in the accumulators represents the hash function's final result. At this point, both parties possess a consistent key string, completing the privacy amplification.

For a Toeplitz matrix based on LFSR, if they utilize a sequential structure to calculate the final results, the required time period is $k \times m$. However, with parallel computing structures, the time period is reduced to m. In Figure 4.5, they illustrate a specific parallel hardware structure for computing the hash function value of a Toeplitz matrix based on LFSR. Since the final secret key is derived from the multiplication of the Toeplitz matrix and the reconciled key, each bit of the reconciled key corresponds to a column of the Toeplitz matrix in the correct order, and each bit of the final secret key corresponds to a row of the Toeplitz matrix sequentially. Every update of the LFSR state is treated as a loop. During each loop, $S_{k-1}, S_{k-2}, ..., S_1, S_0$ store the current LFSR state, equivalent to the shifted column of the Toeplitz matrix. Whether the n bits $S_{k-1}, S_{k-2}, ..., S_1, S_0$ of the LFSR state are individually added to the corresponding accumulators $h[M]_{k-1}, h[M]_{k-2}, ..., h[M]_1, h[M]_0$ depends on the value of the corresponding reconciled key bit M_j. After m loops, the final value in the accumulators constitutes the final secret key sequence. By computing the result of equation (4.3) using this method, the final secret key can be obtained, concluding the privacy amplification process.

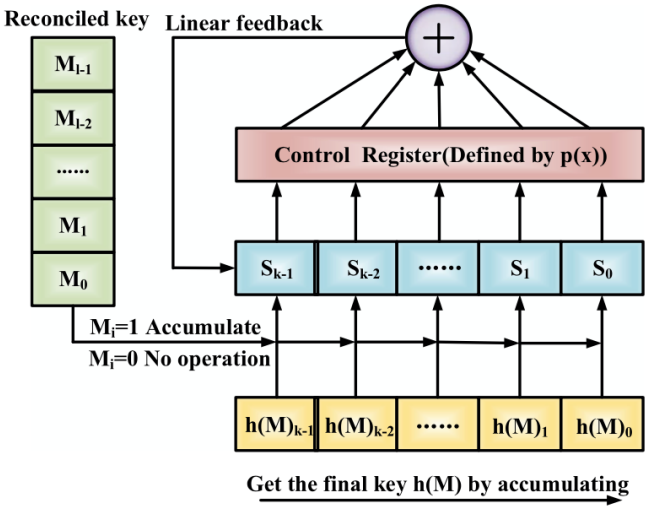


Figure 4.5. Schematic diagram of LFSR-based Toeplitz matrix hash function. (source: [66]).

In this setup, they continue with the example provided at the end of the previous section to illustrate the above procedure. According to the results obtained previously, the first LFSR state is 1,0,0,1, the second LFSR state is 0,1,0,0, the third is 1,0,1,0, the fourth is 1,1,0,1, the fifth is 1,1,1,0, and the sixth is 0,1,1,1. Let the reconciled key be 1,0,0,1,1,0. Initially, the values of the 4 accumulators $h[M]_0, h[M]_1, h[M]_2, h[M]_3$ are all 0. Given that the first bit of the reconciled key is 1, its corresponding LFSR state (the first LFSR state) is added to the accumulators. Consequently, the values of the accumulators become $h[M]_0 = 0+1 = 1$, $h[M]_1 = 0+0 = 0$, $h[M]_2 = 0+0 = 0$, $h[M]_3 = 0+1 = 1$. The second and third bits of the reconciled key are 0, so the second and third LFSR states are not added to the accumulators and thus do not contribute to the calculation of the final secret key. The fourth bit of the reconciled key is 1, causing the fourth LFSR state to be added to the accumulators, updating them to $h[M]_0 = 1+1 = 0$, $h[M]_1 = 0+1 = 1$, $h[M]_2 = 0+0 = 0$, $h[M]_3 = 1+1 = 0$. Similarly, the fifth bit of the reconciled key is 1, resulting in the fifth LFSR state being included in the accumulators. Thus, they obtain $h[M]_0 = 0+1 = 1$, $h[M]_1 = 1+1 = 0$, $h[M]_2 = 0+1 = 1$, $h[M]_3 = 0+0 = 0$. Finally, the sixth bit of the reconciled key is 0, so the sixth LFSR state is not included in the accumulators. Following this process, the final values of the accumulators are $h[M]_0 = 1$, $h[M]_1 = 0$, $h[M]_2 = 1$, $h[M]_3 = 0$, resulting in a final secret key of 1,0,1,0.

4.4 Cellular Automata

In [20] they present a promising PA protocol based on Cellular Automata. Cellular Automata (CA) represents a distinct grid-based dynamic model characterized by discrete attributes in time, space, and state. The state alterations are governed by local rules in either time or space dimensions. CA serves as a general term encompassing a particular model or framework [73, 74]. It is defined as a dynamic system that undergoes discontinuous changes in the time dimension within a unit space comprised of finite, discontinuous elements, all under specific rules. In detail, CA comprises four fundamental components: cell space, state, neighborhood, and rule, denoted as $A = (Ld, S, N, f)$ [75]. Here, A signifies CA, Ld represents cell space, d is the spatial dimension, S represents the finite discrete state set of CA, N is the neighborhood vector, and f is the local conversion function.

4.4.1 Elementary Cellular Automata

Elementary Cellular Automata (ECA) represents the most basic form of CA [76]. It has a state number of $k = 2$, a neighborhood radius of $r = 1$, and the local transformation function f is expressed as

$$s_i^{t+1} = f(s_{i-1}^t s_i^t s_{i+1}^t)$$

The local conversion function takes three state quantities as input, and each state quantity can assume two values, 0 or 1, resulting in eight possible state combination modes: $000, 001, 010, 011, 100, 101, 110,$ and 111. Each input state combination corresponds to two output states, either 0 or 1. Determining the corresponding output for each input state combination yields the truth table of CA, which aligns with the rules of an ECA. Given there are 8 state combinations in ECA, each corresponding to two outputs, there are a total of $2^8 = 256$ truth tables and 256 rules. The rule space is an aggregation of these 256 rules. Serial numbers are assigned to these 256 combinations, and the 8-bit binary number in the right column of each combination table is recorded as a decimal number to obtain the rule number, which is any integer between 0 and 255. The eight possible combination modes are arranged in binary increasing order, and the corresponding output state is calculated simultaneously to derive the truth table of the local conversion function. Table 1 illustrates the truth table of rule No. 150 (the binary representation of 150 is 10010110).

Taking an ECA with a length of 8 bits as an illustration, if the initial value of CA is set to 10101010, and the rules adopted are akin to rule No. 150 as shown in Table 1, the CA is updated as follows: in the first clock cycle, the initial value is assigned to the CA. In the second clock cycle, the 8-bit CA is updated according to the prescribed rules. The neighborhood objects of the first bit of the CA are bit 8 and bit 2, i.e., $s_1^2 = f(s_8^1 s_1^1 s_2^1)$. The resulting status is 010 which

is then updated to 0 based on the truth table. The second bit is updated similarly to get the status 101, which is updated to 1 according to the truth table. The third bit adopts the status 010 and updates it to 0. The fourth to eighth bits adopt $101, 010, 101, 010$, and 101 respectively, and update to $1, 0, 1, 0$, and 1. Ultimately, after a round of updating, the CA with 8 bits attains a new state: 01010101.

$S_{i-1}^t s_i^t s_{i+1}^t$	s_i^{t+1}
000	1
001	0
010	0
011	1
100	0
101	1
110	1
111	0

Table 4.1. Truth table of ECA (No. 150).

4.4.2 Pseudorandom Sequence

True randomness is a phenomenon devoid of ascertainable cause and effect, where we perceive only the result without the ability to observe (i.e., within the existing human cognitive system) the cause. It remains entirely incomprehensible. Currently, the randomness extensively utilized across various domains is typically derived from chaotic systems, like the Duffing oscillator [77]. Chaos represents a phenomenon governed by cause and effect, yet defies precise mathematical calculation and prediction. This is due to the sensitivity of initial conditions and the inherent complexity of the model, placing it within a realm of understandable but imprecise predictive control. Authentic random sequences can only originate from natural phenomena, a challenging task to achieve in practical applications. Consequently, pseudorandom sequences generated through artificial methods are widely employed in the field of sequence cipher [78, 79]. The core challenge in sequence cipher lies in producing a lengthy, unpredictable key sequence. Pseudorandom number generation via Cellular Automata (CA) has been a vibrant area of cryptographic research. One of the driving motivations behind this, stems from the benefits CAs offer, especially from a VLSI (Very Large Scale Integration) perspective: CAs are simple, regular, locally interconnected, and modular [80]. These features render them more hardware-friendly compared to other models. Pseudorandom sequences generated by CA can be categorized into three main types.

- Stationary type: Irrespective of the initial value of CA, after a certain evolution period, it settles into a stationary state where all cell state values are identical. The evolution of this CA type lacks randomness.
- Periodic type: The CA adopts a periodic structure after a specific duration, where the evolution removes some randomness but retains a certain amount. This type finds applications in image processing.
- Chaotic type: The CA assumes a random or chaotic aperiodic state following a specific evolution period. This CA type exhibits strong randomness in its evolution.

In this paper, the pseudorandom sequence generated by CA aligns with the chaotic type. For Elementary Cellular Automata (ECA), a chaotic pseudorandom sequence can be generated based on the truth table rule No. 150. When chaotic CA is selected, there's an absence of discernible regular patterns in the space-time pattern, presenting a much richer pattern compared to a single CA. Leveraging this characteristic, a pseudorandom sequence with superior performance can be generated. The pseudorandom sequence generator using CA necessitates assigning an initial value to the CA before commencing its operation. The initial value of the first CA of length N can either be randomly generated or fixed according to specific applications. For instance, in hash function applications, selecting the first N bits of irrational numbers like e and π, or the first N bits of $\sqrt{2}$ and $\sqrt{3}$ could be employed.

4.4.3 Proposed Algorithm

Regardless of the designer's optimizations, Privacy Amplification (PA) based on the Toeplitz matrix must find a delicate balance between resource usage and time consumption. This balance is significantly influenced by the Toeplitz matrix required by the compression function. Hence, the designers utilize Cellular Automata (CA), a tool known for generating pseudorandom sequences with strong randomness, to replace the Toeplitz matrix. This replacement facilitates the compression process from the negotiation key to the final key, thereby enhancing the speed of the PA algorithm. Building upon this concept, they introduce a high-speed PA algorithm with memory efficiency utilizing CA.

Notation	Definition
T	Negotiation key
n	Length of negotiation key
T_{Mi}	The i-th group negotiation key
M	Group length of negotiation key
K	Number of groups negotiating key
N_j	The j-th block negotiation key
N	Length of Cellular Automata
C	Reciprocal of key compression rate and number of blocks
H_i	Final key obtained by group i
H	Final key
m	Length of zero complement in negotiation key T

Table 4.2. Notations.

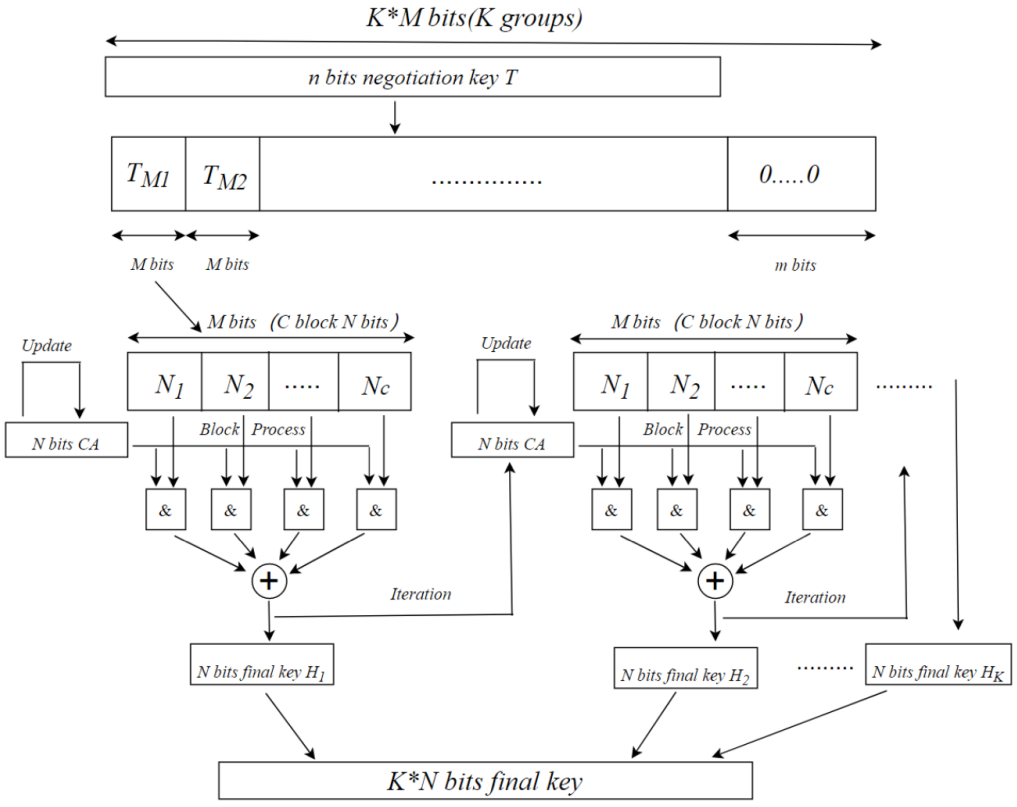


Figure 4.6. Schematic diagram of PA algorithm. (source: [20]).

As depicted in Figure 4.6, the n-bit negotiation key T is initially divided into K groups

denoted as $[T_{M1}, T_{M2}, ..., T_{MK}]$, each having a length of M. Once the negotiation key is divided, each group is sequentially processed. For instance, considering the group negotiation key T_{M1} with a length of M, it is further divided into C blocks, each having a length of N. To ensure that M aligns with $M = C \times N$, it is essential to append a zero sequence of $m = M \times K - n$ bits after the original negotiation key T. At the core of this algorithm lies the utilization of Cellular Automata (CA) to concurrently generate multiple pseudorandom sequences possessing strong randomness. These sequences are then effectively compressed from the group negotiation key of length M into the final key H_1 of length N through a specific operation. Once the final key result is obtained, the algorithm employs an iterative approach, considering the current N bits final key H_1 as part of the process. As the initial value of CA for the subsequent group, the same algorithm is employed to obtain the next N bits of the final key. These steps are repeated until all the negotiation key groups are processed, resulting in the final required $K \times N$ bits of the final key denoted as H. Figure 4.7 illustrates the processing flow of the algorithm. The algorithm's specific steps are detailed below.

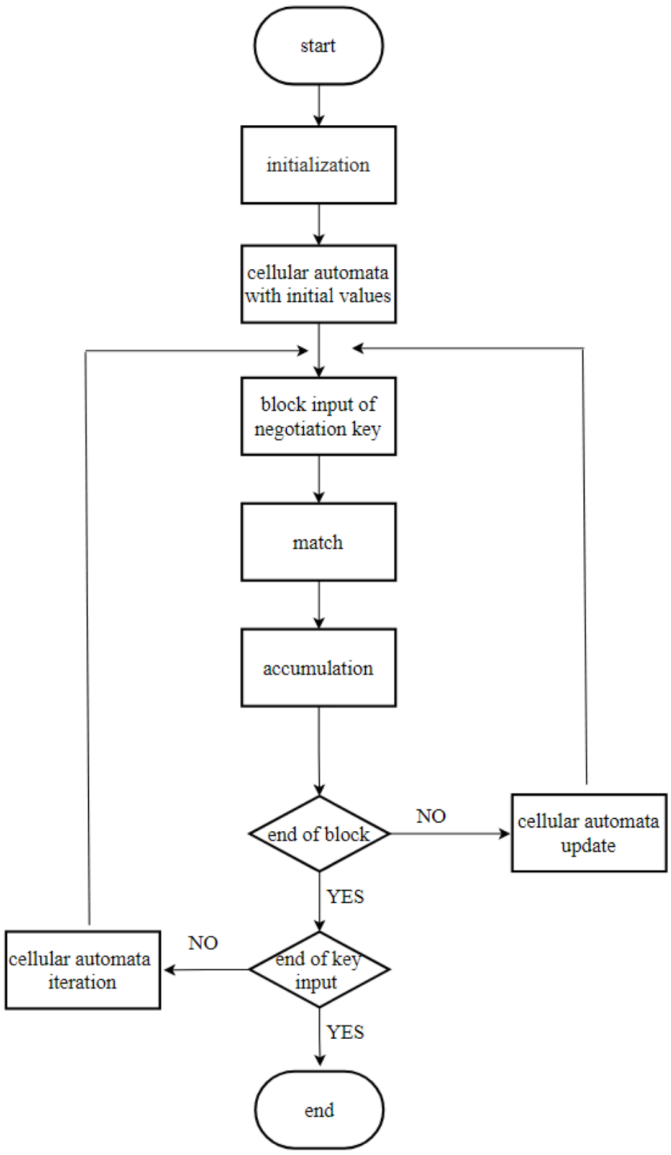


Figure 4.7. Flow chart of PA algorithm. (source: [20]).

Step 1: Set the Parameters According to the Requirements Set parameters such as the length N of CA and the reciprocal C of the compression rate of the final key. Divide the

received n-bit negotiation key T into smaller groups of length M. The last group T_{Mk} might not meet the length requirement. If it doesn't, a sufficient number of zeros, denoted as m, are added to the last group to meet the length requirement.

Step 2: Initialize CA and Set its Running Rules To ensure the pseudorandom sequence generated by CA exhibits strong randomness, appropriate rules need to be chosen. Rule 150, as shown in Table 1 among the 256 rules of ECA, is selected. Under this rule, the CA-generated sequence's space-time map demonstrates evident chaotic characteristics. Regarding the choice of CA's initial value, the fixed first N bits of e, π, $\sqrt{2}$, or $\sqrt{3}$ are chosen.

Step 3: Further Divide the Group Negotiation Key T_M1 Divide the group negotiation key $TM1$ with length M into blocks of length N. Combine these blocks with the N-bit sequence generated by CA through a bitwise AND operation. Specifically, the first N length block T_{M11} is combined with the initial value of the CA denoted as $s_i^1, i = 1,2, ..., N$, and the result is placed into the N-bit accumulator. After the first block operation, utilizing CA's characteristics, they can simultaneously update the N-bit data of the CA. The updated result, denoted as $s_i^2, i = 1, 2, ..., N$, is subject to a bitwise AND operation with the negotiation key of the next N length block T_{M12}. The result is also placed into the N-bit accumulator and undergoes modulo-2 addition with the previous result. These steps are repeated until the C blocks of M length negotiation key are calculated, using the value of the accumulator as the final key H_1. Embracing the idea of iteration, the result of the final key H_1 is taken as the N-bit initial value of the next CA. The above process is reiterated, completing the calculation of the last group to obtain the final key H_K.

Step 4: Obtain the Final Key with K Blocks After calculating all the negotiation keys, the final key with K blocks of length N is obtained and combined into a final security key H. Exploiting the fact that the final key is obtained in blocks, the algorithm can output the final result of the privacy amplification process in real time, enhancing the data throughput of the hardware implementation.

In summary, the proposed PA algorithm follows these steps.

$$n + m = K \times M$$

$$H_0 \leftarrow IV$$

$$H_j = g(T_{MJ}, H_{j-1}) = \bigoplus_{l=1}^{C} T_{Mjl} \& a_l, j = 1,2, ..., K$$

$$a_1 = (s_1^1, s_2^1, ..., s_N^1), l = 1,2,3, ..., C$$

$$s_i^l = f(s_{i-1}^{l-1}, s_i^{l-1}, s_{i+1}^{l-1}), s_0^{l-1} = s_N^{l-1}, s_{N+1}^{l-1} = s_1^{l-1}, l \neq 1, i = 1.., N$$

$$H = \{H_1, H_2, ..., H_k\}$$

Definition A family of hash functions is called $\epsilon - balanced$ if $\forall T \neq 0, c, Prh(h(T) = c) \leq \epsilon$.

Theorem For any values of$N$and M the above defined family of hash functions is $\epsilon - balanced$ for $\epsilon \leq \frac{1}{2^N}$. **Proof** To show that the family is $\epsilon - balanced$, notice that any non-zero message T_{Mj} of length M and any string c of length $N, h(T_{Mj}) = c$ iff $\bigoplus_{l=1}^{C} T_{Mjl} \& a_l = c$. Since a_l generated by ECA under rule 150 has random characteristics, the vector T_{Mj} assumes this value c with probability of $\frac{1}{2^N}$, and therefore $Pr(h(T_{Mj}) = c)$ happens with at most this probability.

Chapter 5

Resource-efficient implementation of QKD post-processing

In this chapter we will analyze the presented EC protocols and PA algorithms, form a resource-efficient perspective, excluding the possibility of hardware acceleration, but focusing instead on a more low-resources hardware. In the following section we will

5.1 Efficiency Evaluation of Error Correction Protocols

Imagine a covert communication channel between two individuals, Alice and Bob. In this setup, Alice transmits an n-bit string A to Bob. Now, let's introduce some technical elements into this scenario. We model this clandestine channel using what's known as a Binary Symmetric Channel ($BSC(p)$), where p represents the probability of an error occurring during transmission. After traversing this potentially error-prone path, Bob receives the transmitted data as an n-bit string, which we denote as B.

Within this context, both Alice and Bob have a common objective: they want to create a shared n-bit secret string, which we'll refer to as S. The source of S is the n-bit strings A and B that have been transmitted. It's essential to highlight that this process takes place over a public channel, and notably, this public channel is free from interference, a critical assumption. Importantly, no other security assumptions are being made at this stage.

Now, why is this process significant? The primary goal here is to design protocols that minimize the information leakage about the secret string S. This is crucial because there's a possibility of an eavesdropper, a highly computationally capable one, who may be eavesdropping on the public channel. The lesser information they can extract about S, the more secure the communication.

In the realm of Error Correction (EC) protocols, it's paramount to define what these protocols are. An Error Correction protocol, denoted as EC, is essentially a product of the algorithms employed by Alice and Bob. When these algorithms operate on the strings A and B, their collective objective is to generate the string S. This generation process involves the exchange of certain information, which we represent as Q, over the public channel. In simpler terms, this whole operation is encapsulated by the notation $EC = [S, Q]$ or $EC(A, B) = [S, Q]$, which is context-dependent.

Nevertheless, not all EC protocols must aim for the utmost level of optimality. Prior to executing the protocol, Alice and Bob can mutually decide to reveal a slight surplus of information concerning the theoretical limit. This additional information becomes negotiable. What's crucial here is the practicality of the protocol. The ultimate aim is an efficient protocol that aligns with the operational needs. A real-world example would be in cases where the bits exchanged over the secret channel are particularly precious. In such scenarios, it might be more prudent to allocate additional computational resources during the correction process, thereby preserving these valuable bits.

5.1.1 Winnow Protocol

In the year 2003, a novel protocol called Winnow, which hinged on the principles of Hamming codes, made its debut in the field of secure communication [25]. The primary objective behind this introduction was to boost data throughput while simultaneously reducing the interaction requirements of a previously established protocol known as Cascade. The key innovation here was the elimination of a complex binary search step from the process.

In a Winnow-based exchange, the two principal parties, Alice and Bob, embark on a rather organized data division. They break down their respective random keys, labeled as M_a and M_b, into manageable blocks of equal length. It's advisable to start with a recommended block size of $k = 8$ for efficient operation. Now, the magic happens when they conjure what are known as syndrome values, S_a and S_b. These are computed through a process involving two vital matrices: a Generator matrix, G, and a parity check Matrix, H, in which the equation $H \cdot G^T = 0$ plays a pivotal role.

Let's dive into the details of how the Winnow protocol unfolds. At its core, this method operates on blocks of data, each of size k. Bob leverages his share of the key, M_b, to generate and transmit a syndrome, denoted as $S_b = H \cdot M_b$, to Alice. Once this syndrome arrives, Alice conducts an intricate analysis, focusing on the syndrome differences, S_d. If these differences are non-zero, it indicates the presence of errors. In response, Alice undertakes error correction, striving to minimize alterations to the data. Her primary aim is to ensure that the syndrome values ultimately converge to zero.

However, it's worth noting a significant limitation of the Winnow protocol, primarily stemming from its reliance on Hamming codes. This protocol, although innovative, can sometimes introduce errors during its operation. This issue marks one of its principal shortcomings. In practice, its efficiency tends to be lower when compared to Cascade, especially when dealing with Quantum Bit Error Rate (QBER) values below 10%. These lower QBER values are particularly relevant for practical Quantum Key Distribution (QKD) scenarios.

To address this limitation and maintain information-theoretical secrecy, an approach was suggested by Buttler. It involves the intentional omission of an extra bit from each block of size k during the privacy maintenance phase. This strategic adjustment contributes to reinforcing the security of the Winnow protocol, despite its occasional error introduction.

5.1.2 LDPC Protocol

When dealing with terrestrial communication links, Alice and Bob typically enjoy the luxury of time, flexibility in computation, and minimal constraints regarding communication complexity. However, the scenario shifts dramatically when satellite links come into play. Here, various challenges arise, such as substantial channel losses, time limitations imposed by periodic satellite passages, and heightened demands on both computational and communication resources.

In response to these challenges, recent years have witnessed a growing interest in the application of Gallager's Low Density Parity Check (LDPC) codes. These codes have demonstrated remarkable error correction capabilities at rates surpassing those of the Cascade and Winnow protocols.

One of the key advantages of LDPC lies in its efficiency regarding communication overhead. Moreover, it naturally accommodates the asymmetric distribution of computational power required at each end of the communication channel. These properties make LDPC a favorable choice for satellite-based quantum communication, where resource constraints loom large.

LDPC codes revolve around two essential matrices: the parity check matrix, denoted as H, and the generator matrix, referred to as G. The effectiveness of an LDPC code is characterized by its minimum distance, which is used to define the decoding limit of the code.

The dimensions of these matrices are defined as $m \times n$, with m given by the formula $m = n \cdot (1 - r)$, where r represents the code rate and falls within the range of $[0, 1]$. The code rate, typically predetermined, governs the code's error correction capabilities and efficiency.

The reconciliation process in the LDPC protocol unfolds as follows:

1. QBER Estimation: The first step involves estimating the Quantum Bit Error Rate (QBER) of the communication channel.

2. Matrix Selection: Based on the estimated QBER, Alice and Bob collaboratively select identical $m \times n$ generator matrix G and parity check matrix H configurations.

3. Syndrome Calculation: For every sifted key, Bob calculates a syndrome, S_b, and transmits it to Alice.

4. Error Correction: With the syndrome in hand, Alice endeavors to reconcile the sifted key. Her primary objective is to reconstruct Bob's key vector, labeled as x. To achieve this, she leverages her own key vector, y, the received syndrome S_b, the parity-check matrix H, and the estimated QBER value. Alice can employ various decoding techniques for LDPC, such as the belief propagation decoding algorithm (also known as the Sum-Product algorithm) or Log-Likelihood Ratios. These techniques significantly reduce computational complexity.

However, it's essential to acknowledge that decoding LDPC codes typically demands more extensive computational resources and memory compared to the Cascade or Winnow protocols. Despite this drawback, LDPC offers a compelling advantage by minimizing the demand for communication resources. This becomes particularly valuable in networks constrained by limited bandwidth and latency.

In the realm of Quantum Key Distribution (QKD), LDPC found its initial application as the foundation for the BBN Niagara protocol within the DARPA QKD network [29].

5.1.3 Cascade

The Cascade protocol operates on the principle of employing a binary search mechanism to pinpoint and rectify erroneous bits. This binary search process entails further subdividing the data block into two smaller subblocks, comparing the results of parity checks until the erroneous bit is identified. In situations where a block contains an erroneous bit, a total of $1 + [\log_2 k_i]$ parity values are exchanged. Here, $1 + [\log_2 k_i]$ represents the maximum number of times that a block, denoted as k_i, can be partitioned. In cases where the blocks are error-free, only a single parity value is exchanged.

As an additional security measure, it is recommended to eliminate specific bits. These bits include a portion of the sample used to estimate the Quantum Bit Error Rate (QBER) and, crucially, the last bit of each block and subblock where the parity bit was exchanged. This practice significantly reduces the volume of information that could be acquired by a potential eavesdropper, often referred to as Eve.

The protocol introduces the concept of the maximum number of discarded bits, denoted as D_i, which can be calculated based on the value of k_i in the i^{th} iteration. This value depends on the QBER and the initial block size. The total number of discarded bits, represented as D, can be computed as follows:

$$D = \sum D_i = \sum_i \left(\frac{n}{k_i} + \sum_{\text{errors corrected}} [\log_2 k_i] \right)$$

Where $k_i = 2 \cdot k_{i-1}$ and $k_i < \frac{n}{2}$, with n denoting the total number of measured values in the sifting phase.

It's important to note that the quantity of discarded bits is contingent upon the QBER value and the initial block size. Over the years, numerous reviews and enhancements have been proposed for Cascade protocols, making the Cascade approach highly adaptable and customizable to cater to various application scenarios.

5.1.4 Conclusions on EC algorithm choice

In [26], the primary Error Correction (EC) protocols introduced in chapter 3 are subjected to a comparative analysis based on the metrics elucidated within the same chapter. It's imperative to remember that protocol efficiency exhibits an inverse relationship with the number of leaked bits, as emphasized in the context of the Cascade protocol [24].

Number of Leaked Bits

The total number of leaked bits is quantified as follows for the considered EC protocols:

- Cascade: With each exchange of parity values, one bit is discarded.
- Winnow: For each block denoted as k, one bit is discarded.
- LDPC: The total length of the syndrome S_b exchanged.

The number of leaked bits represents the foremost factor to contemplate when assessing an EC protocol. As discussed in chapter 3, the number of leaked bits plays a pivotal role in...

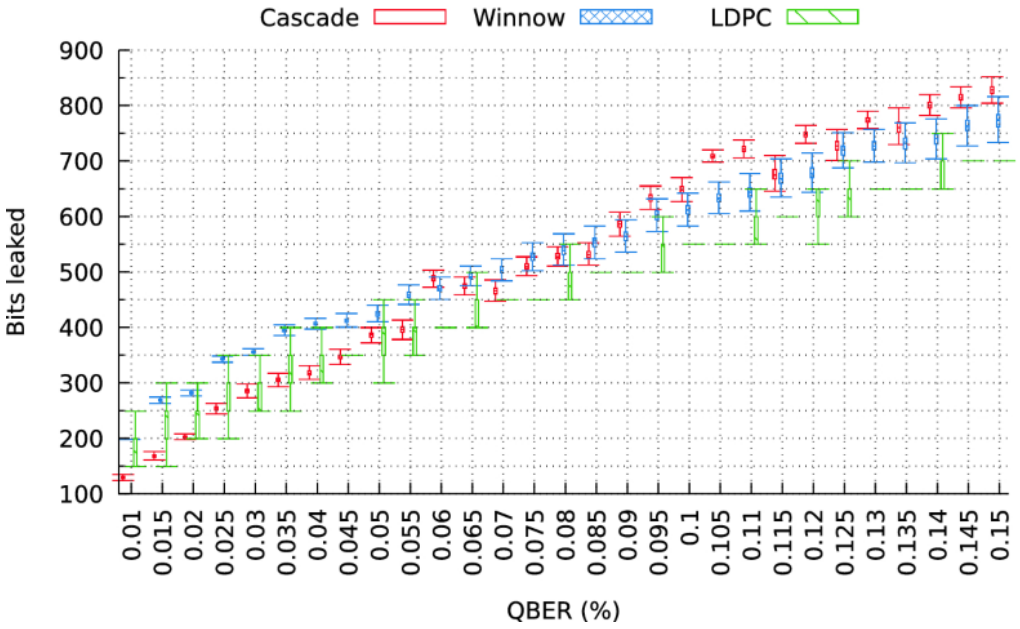


Figure 5.1. Comparison of EC protocols number of bits leaked (source: [26]).

Figure 5.1 visually demonstrates that for lower Quantum Bit Error Rate (QBER) values (up to 0.05%), the Cascade protocol adeptly identifies and rectifies errors, resulting in a limited number of iterations. However, as the QBER value escalates (up to 0.10%), the LDPC protocol emerges as more efficient in terms of overhead and exchanged information. Nonetheless, it's worth noting that the disparity in efficiency is relatively marginal and primarily applicable to a specific range of QBER values (9-15%) that scarcely intersects with the acceptable range (0-11%).

Execution Time

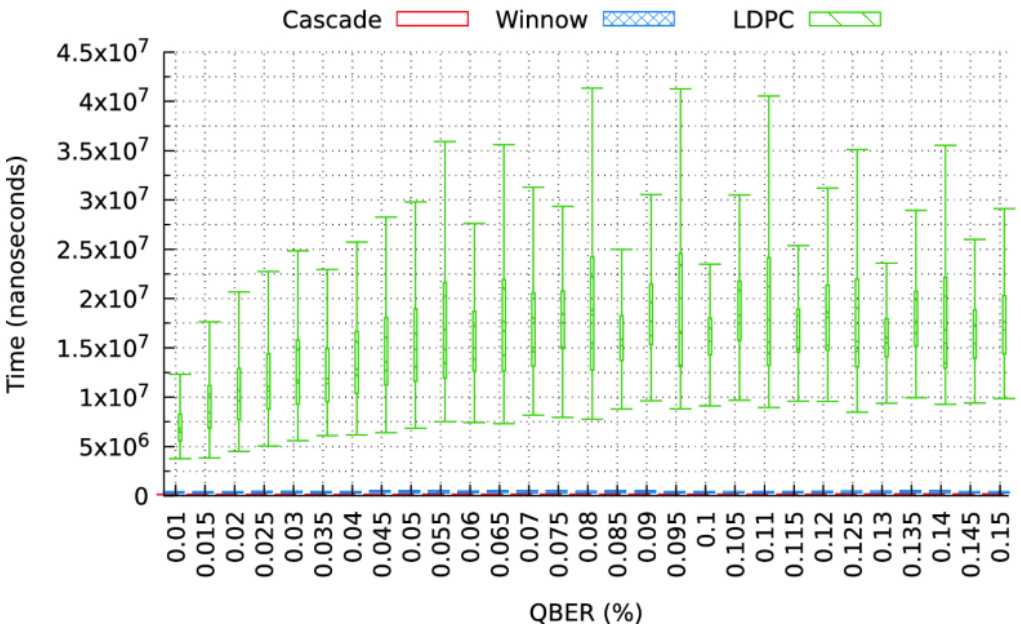


Figure 5.2. Comparison of EC protocols execution time (source: [26]).

Figure 5.2 illustrates that efficiency in terms of overhead exacts a price in execution time. Owing to the simplicity of their algorithms, both Cascade and Winnow protocols boast nearly constant execution times. In contrast, the execution time of LDPC varies, gradually increasing with an escalation in QBER. It's important to note that LDPC utilizes the belief propagation algorithm for decoding.

In conclusion, the preference for EC protocols leans toward Cascade. This preference is attributed to Cascade's efficiency, not only in terms of resource utilization but also concerning better performance compared to Winnow, especially in scenarios featuring realistic QBER values.

5.2 Privacy amplification protocols resource-efficiency comparison

5.2.1 FFT

Within this section, the evaluation of the High-Speed and Large-Scale (HiLS) Privacy Amplification (PA) scheme is analyzed with a primary focus on resource efficiency. It is of paramount importance to emphasize that the precision of Fast Fourier Transform (FFT) operations may be constrained due to finite-precision floating-point arithmetic. To ensure the highest efficiency and address synchronization and thread safety concerns, shared memory multi-processes are employed for carrying out various calculations, including Fourier transforms and Hadamard products.

The throughput of the HiLS PA scheme is meticulously scrutinised through a comprehensive evaluation involving different input scales (n) and sub-block sizes (m). What becomes evident is the HiLS PA scheme consistently attains the highest throughput performance when the splitting factor mn assumes a value of 0.125. However, it is notable that when mn falls below 0.0625 or exceeds 0.25, the scheme experiences reduced performance, resulting in suboptimal throughput. For instance, with an input scale of 512 Megabits per second (Mbps), the HiLS PA scheme's optimized throughput is recorded at 59.06 Mbps, 50.48 Mbps, and 30.49 Mbps for compression ratios of 0.125, 0.25, and 0.50, respectively.

A critical observation emerges from the simulation results: for 10 Gigahertz (GHz) entanglement-based Quantum Key Distribution (QKD) systems, the maximum compression ratio required for PA schemes is 0.297. These results are pivotal as they inform the performance evaluation of the HiLS PA scheme against various implementations on different platforms, such as FPGA and CPU platforms. Strikingly, the HiLS PA scheme's throughput outpaces several prior works. However, it is crucial to underline that for extremely large inputs, the implementation's computational resources play a significant role in the achieved throughput. With limited resources, HiLS PA scheme's throughput can drop to 0.44 Mbps for an input scale of 128 Gigabits per second (Gbps).

This critical analysis underscores that the HiLS PA scheme is an efficient and adaptable solution, especially when applied to high-speed QKD systems with larger input scales. The study emphasizes that even with limited computational resources, the scheme performs remarkably well. Furthermore, the paper points out that potential acceleration avenues could be explored, especially through the integration of hardware acceleration methods like FPGA and GPU, further enhancing the scheme's performance. Of course this kind of hardware enhancing is outside the scope of this work and thus not analyzed.

5.2.2 LSFR

This subsection offers an in-depth examination of the implementation of a Linear Feedback Shift Register (LFSR) in a privacy amplification algorithm, a crucial step in the realm of Quantum Key Distribution (QKD). LFSR is introduced as a groundbreaking solution designed to conserve storage space, optimize the speed of privacy amplification, and uphold the stability of Continuous Variable Quantum Key Distribution (CVQKD) systems. The innovative use of LFSR enables the efficient construction of Toeplitz matrices for privacy amplification, revolutionizing the field.

The paper delves into the intricacies of the LFSR-based algorithm and subjects it to rigorous simulations. The findings are nothing short of remarkable; under identical hardware conditions, the LFSR algorithm significantly outperforms its non-LFSR-based counterpart, reducing time consumption to nearly half. This revelation underscores the immense efficiency gains of the LFSR algorithm, particularly when privacy amplification is applied to scenarios requiring high-speed data processing. Additionally, the results highlight that the LFSR algorithm not only conserves valuable memory space but also accelerates the privacy amplification process substantially.

Furthermore, the paper posits that when LFSR is synergistically integrated with other hardware acceleration methods, such as Field-Programmable Gate Arrays (FPGA) and Graphics Processing Units (GPU), the speed of privacy amplification could be further augmented. This integration is expected to be particularly advantageous when dealing with large data packets, opening new frontiers in high-speed privacy amplification.

5.2.3 CA

In the pursuit of advancing privacy amplification techniques, the paper presents a compelling exploration of the Cellular Automata (CA) privacy amplification scheme. A rigorous performance comparison is undertaken, allowing us to gain a deeper understanding of its merits, especially when juxtaposed with other methods under various experimental conditions. This approach aims to bolster the security and efficiency of privacy amplification processes, which is of critical importance in the context of modern cryptography.

First and foremost, the paper carefully sets the stage by configuring different negotiation key lengths ranging from 0.64 million bits to 5.12 million bits. The intent is to scrutinize the performance of the CA algorithm in various scenarios. The outcomes are unequivocal, as under the stipulated experimental conditions, the CA algorithm consistently exhibits significantly reduced time consumption compared to the block LFSR and FFT algorithms. This observation underscores the CA algorithm's superiority in terms of key generation rates and overall algorithm execution speed.

Notably, the research explores the impact of different compression ratios on these algorithms, revealing that the compression rate, while influential, has a relatively minor effect on execution

time. This discovery underscores the robustness of these algorithms, especially the CA-based approach, which proves to be less sensitive to alterations in compression ratios.

In summary, the CA-based privacy amplification scheme emerges as a robust and versatile solution. Its adaptable nature, combined with its efficiency, renders it a powerful tool in the realm of modern cryptography, capable of enhancing both the speed and security of privacy amplification. The paper concludes by underscoring the potential for combining the CA approach with other hardware acceleration methods, while maintaining the same security standard of the other proposed algorithms.

5.2.4 Conclusions on PA algorithm choice

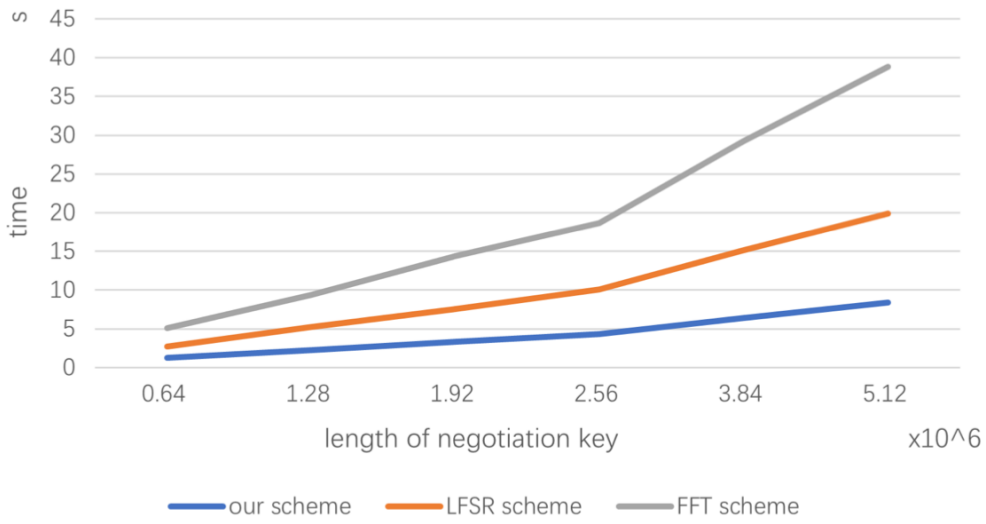


Figure 5.3. Comparison of PA protocols execution times (source: [20])

In figure 5.3 the presented PA algorithms are compared following the metric proposed in chapter 4, the running time. The graph [20] shows that the Cellular Automata algorithm has better performance, compared to the other two. It is fair to notice that both the LSFR [66] and FFT [50] propose some hardware acceleration methods that could speed up their algorithms to better performances, but as mentioned above, the argument of hardware acceleration is beyond the scope of this work. In the following section we'll present some implementation choices based on Cellular Automata.

5.3 QKD post-processing application

The proposed solution creates a QKD-post processing module, composed of two sub-modules wrapped together, the Error Correction and Privacy Amplification modules. The software is designed to be used as a stand-alone application or inside a QKD simulator, in both cases it required a softwarized network.

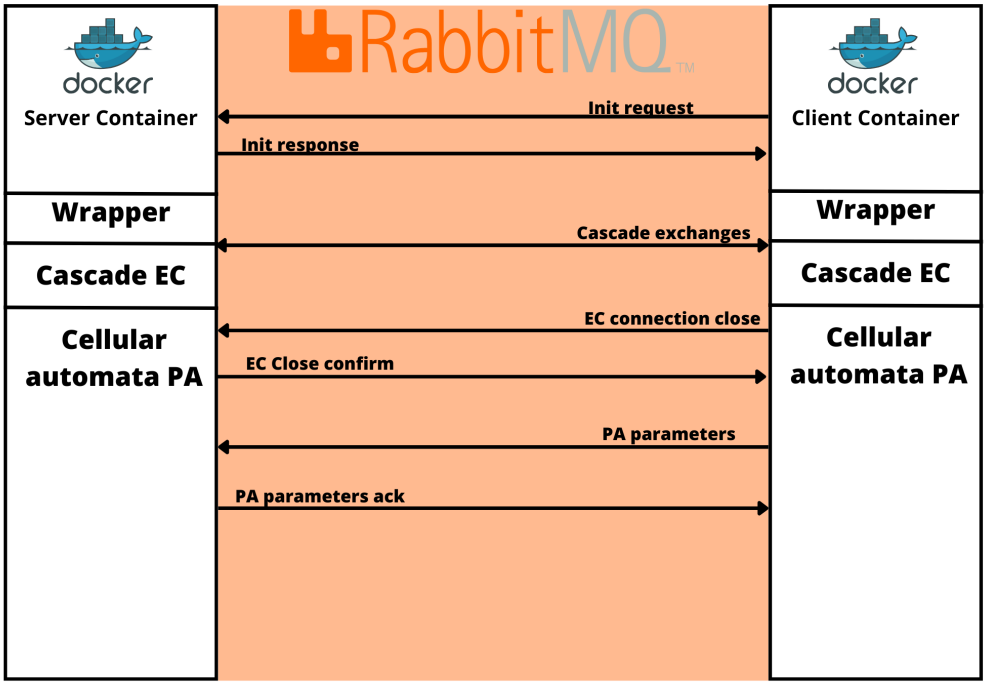


Figure 5.4. Application scheme and exchanges.

5.3.1 Design and Architecture

The general structure consists of a Python wrapper that executes two sub-modules for post-processing in series and extracts efficiency and running time data at the end. The purpose of the post-processing software is twofold: first, to implement a resource-efficient solution for Quantum Key Distribution (QKD) post-processing that can be integrated into QKD solutions within software-defined networks, and second, to provide a tool for testing the performance and efficiency of various Error Correction (EC) and Privacy Amplification (PA) protocols.

The software relies on RabbitMQ for message exchange between Alice and Bob. For use in software-defined QKD networks, an active instance of RabbitMQ is necessary as a message broker.

For testing EC and PA protocols, the software automatically builds a RabbitMQ container that it uses during simulations.

The software can be used in various forms. For individual post-processing demos, you can use the HTML interface exposed on localhost by running the DEMO script. For broader testing and implementations in QKD distribution designs, you can utilize the APIs or the RUN_TESTS script.

Rabbitmq

RabbitMQ serves as a message broker, handling the acceptance and forwarding of messages. To understand its function, you can liken it to a post office. When you drop your mail into a postbox, you trust that the letter carrier will eventually deliver it to the intended recipient. In this analogy, RabbitMQ plays the roles of both the postbox and the letter carrier.

However, RabbitMQ differs significantly from a traditional post office in that it deals with digital data in the form of binary messages. It accepts, stores, and forwards these data blobs.

In the context of RabbitMQ and messaging systems in general, some specific terms are used:

- **Producing:** This term is equivalent to sending, where a program that sends messages is called a producer. A producer sends messages to a queue. - **Queue**: In RabbitMQ, a queue

is akin to the postbox mentioned earlier. Messages pass through RabbitMQ, but they can only be stored within a queue. A queue's size is constrained by the host's memory and disk limits, effectively acting as a message buffer. Multiple producers can send messages to a single queue, and multiple consumers can attempt to receive data from that same queue.

It's essential to note that producers, consumers, and brokers do not need to reside on the same host, and in many applications, they won't. An application can even act as both a producer and a consumer.

The producer (labeled "P") sends messages to a queue, which acts as a message buffer managed by RabbitMQ on behalf of the consumer (labeled "C"). The overall design is as follows:

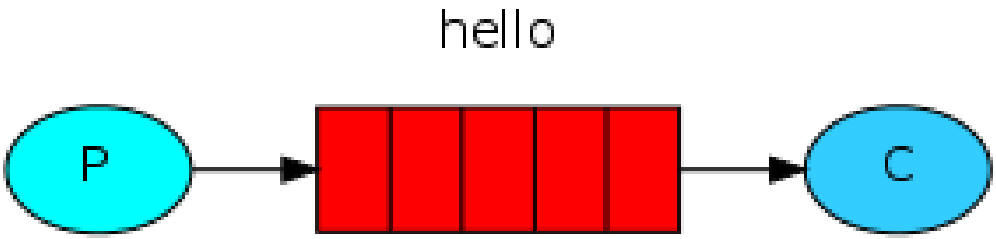


Figure 5.5. Rabbitmq overall design

Producer -¿ Queue -¿ Consumer: Sending and receiving messages from a named queue.

The producer sends messages to the "hello" queue, and the consumer receives messages from that same queue.

To implement RabbitMQ in the proposed solution, the open-source AMQP-CPP library by Copernica Marketing Software [81] has been utilized.

Memory management for key storing

In order to effectively implement a practical application of Elliptic Curve (EC) and Pairing-Based (PA) cryptography, the secure storage of cryptographic keys in memory plays a critical role. Achieving a resource-efficient implementation requires that the key remains unaltered during its storage in memory. In other words, the bits comprising the key must be stored in memory exactly as they are represented in the original key. For this purpose, the solution proposed in this work adopts a key storage system initially introduced by [22], as follows:

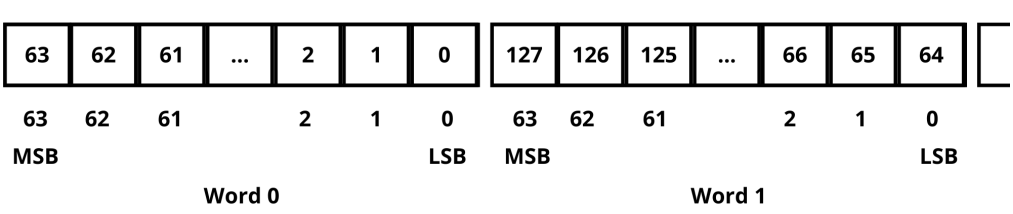


Figure 5.6. Memory managing of key.

The key is stored in memory using vectors of uint64_t, which represent the complete address of a word in memory in 64-bit architectures. It is evident that this approach requires a more

complex management of the key, which is now divided into 64-bit words. For example, when manipulating individual bits, it is necessary to use masks that isolate the specific bit within the word to manipulate it. The same principle applies to groups of bits.

This method allows the key to be manipulated without relying on high-level structures, directly operating on the bits themselves. Achieving the same result would not have been possible using strings or the bitset structures proposed in C++23, which use more complex structures to manage individual entries.1

5.3.2 Error Correction

To develop the EC sub-module i started from [22] as a skeleton. The software implements several variations to the original cascade protocol, the most studied combination of them are available to test or use in the proposed solution. The EC sub-module was developed using C++23 and can used also a stand-alone application to perform tests on Error correction by itself, in the proposed solution is run by the wrapper and all the data is collected from the JSON file that is produced. The implemented parameters are similar to the one presented in chapter 3 with the difference of the possibility to use singleton block or deterministic shuffling of the key. The software by default doe not implement the use of singleton blocks and the key shuffle is always random.

- N° of cascade and BICONF iterations
- Initial Block size
- Block size function (for following iterations)
- Key shuffle caching (block reuse)

Table 5.1. Table of implemented parameters. $\alpha = \log_2(1/QBER) - \frac{1}{2}$

Algorithm	Initial Block size k1	Block size function	Cascade Passes	$BICONF$ passes	Shuffle caching (Block Reuse)
Original	0.73/QBER	$2k_{i-1}$	4	0	Yes
BICONF	0.92/QBER	$3k_{i-1}$	2	10	Yes
Yanetal	0.80/QBER	n/2	10	0	No
Option3	QBER	n/2	16	0	No
Option4	QBER	n/2	16	0	Yes
Option7	$2^{\log_2 1/QBER}$	n/2	14	0	Yes
Option8	2^{α}	n/2	14	0	Yes

Workflow

The process start when the MyCascade software is called and is provided with the number of bits of the key, the endpoint, the QBER estimation, the key if provided (only for the server) and all the data necessary to connect to the running Rabbitmq instance (IP address, port, user and password)

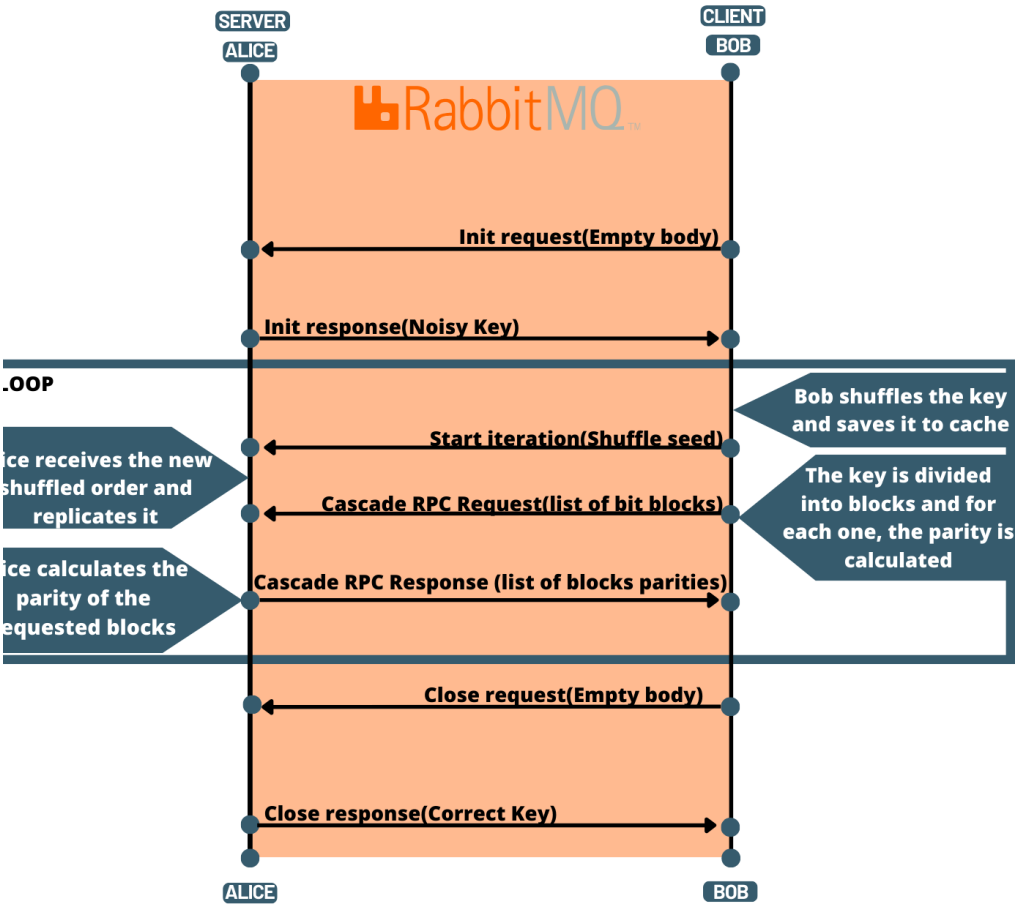


Figure 5.7. Error correction Exchanges.

1. To begin Bob sends a empty body message to serve as an exchange initialization.
2. Alice (Server) replies to bob sending the noisy key (of course this practice is done only for testing purpose). The real purpose of this message is to initiate the exchange, it in fact serves as an exchange initialization.
3. Bob then proceeds by sending to Alice the first shuffle seed, thus starting the iteration. This process will be performed at the start of every iteration.
4. Right after bob sends Alice the first message containing the set of bits block he's checking at the moment.
5. Alice responds with the parity bits of the selected blocks. This exchange takes the form of a Remote Procedure Call in which Bob prompts ALice with some bits indexes (start and end of a block) and Alice responds with the corresponding parities.
6. Bob then proceed by binary searching error in blocks with wrong parity, thus generating sub-blocks and requesting again the parity repeating step 4 and 5
7. Steps 3-5 are repeated until all the iteration are done
8. At the end of all iterations Bobs sends Alice an exchange closing message, which Alice responds by sending the correct key, this, of course is done to evaluate remaining errors.

Bob then calculates the EC performance data and produces a JSON file.

Execution results

The Error Correction module at the end produces a JSON file with all the relevant data from the protocol execution in particular:

- EC running time
- Number of channel uses (round trip)
- EC efficiency
- Exchanged key
- Corrected key
- Remaining errors in the key
- Chosen protocol parameters

In the test implementation, the server will send the correct key via classical channel at the end, clearly this practice is not part of any EC protocol, but in this implementation it is done to check for remaining errors in the key in order to collect data on EC protocols behavior.

Criticalities

This implementation suffers of some issues that will now be discussed.

The first critical issue is related to the use of RabbitMQ, specifically the way data is formatted in RabbitMQ messages. The message formatting requires transmitting key bits in UTF-8 format, converting them from uint64_t. This continuous conversion can sometimes lead to errors. Additionally, the AMQP-CPP protocol lacks fault tolerance, which means that if the messages are not well synchronized, both parties are at risk of encountering network errors that may even result in crashes. Lastly, the absence of authentication during the EC phase is certainly a concern and needs to be addressed before it can be considered suitable for real-world solutions.

5.3.3 Privacy amplification

The PA sub-module was developed using C++23 and can used also a stand-alone application to perform tests on Error correction by itself, in the proposed solution is run by the wrapper and all the data is collected from the JSON file that is produced. The Cellular Automata PA parameters implemented are the same presented in chapter 4:

- K blocks in which the key is divided
- M bits of which each block $T_i, i \in (0, K)$ is composed, note that $M \cdot K$ must be greater of equal to the number of bits of the key
- N bits of the CA

It is important to mention that the user cannot provide all the parameters to the wrapper, when launching a simulation, the user will be able to insert the number k of initial block and the compression ration for PA, the wrapper will then calculate M and N accordingly.

Worlflow

The process start when the PrivacyAmplificationCPP software is called and is provided with the key and it's size, the endpoint, the parameters for Cellular Automata reported below, the key if provided (only for the server) and all the data necessary to connect to the running Rabbitmq instance (IP address, port, user and password). Please note that all the rabbitmq messages acks are automatically delivered when the receiver downloads the message.

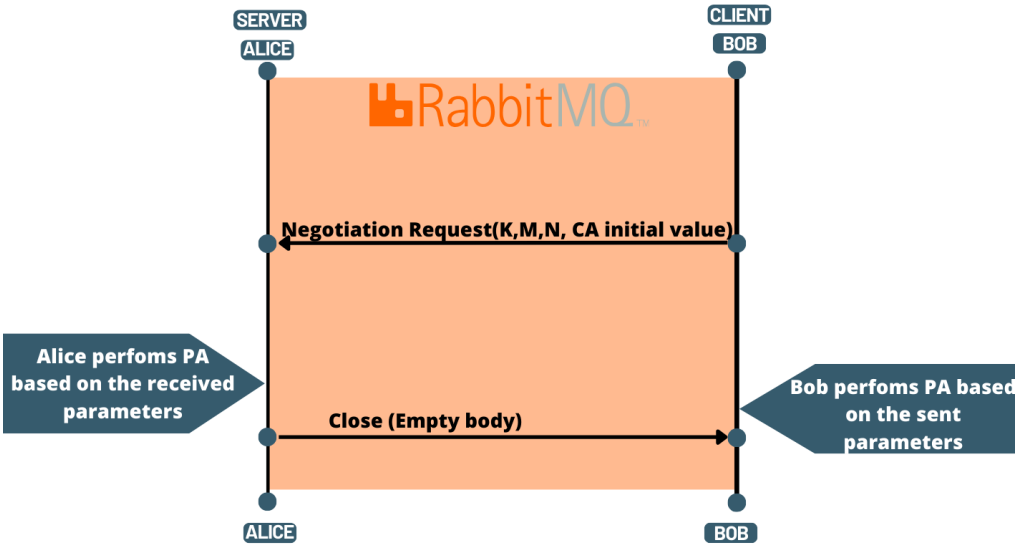


Figure 5.8. Privacy Amplification exchanges.

1. Bob sends Alice the parameters to perform PA , K,M,N and the inital value of the CA, for this implementation the inital value is randomly determined, while in the original design, they suggested to use the first N digits of π or $\sqrt{2}$, but for a simple hardware with large key sizes the calculation could result in a bottleneck for the running time.
2. The key is divided into K blocks of size M to do this the uint64_t vector data is divided in K uint64_t vectors
3. Every K block is divided in C block of size N , creating other uint64_t vector of size N/64 rounded up.
4. An *and* operation is performed for each block with the CA and the CA is updated every time
5. All the resulting blocks are accumulated via a xor operation
6. the result is store as the N^{th} part of the final key
7. the final key is then compressed, eliminating memory gaps between uint64_t words
8. Alice send bob a confimation message to communicate the succesful PA

Execution results

The Privacy Amplification sub-module at the end produces a JSON file with all the relevant data from the protocol execution in particular:

- PA running time

- Exchanged key
- Amplified key
- Chosen protocol parameters

Criticalities

This implementation suffers of some issues that will now be discussed. The first critical issue involves dividing the key, which is stored in a vector of uint64_t, into additional vectors of uint64_t. The total size of these vectors may not necessarily be a multiple of 64 bits. Additionally, using RabbitMQ for parameter transfer can introduce potential synchronization errors between the server and client.

Chapter 6

Testing and results

The proposed solution has undergone extensive testing to gain insights into its limitations and to pinpoint potential enhancements. To replicate a distributed environment, each participating instance in the communication process needed to run on a separate machine. Docker containers were employed to evaluate this setup, and container orchestration was managed via Docker Compose, streamlining management through a single configuration file. All the testing procedures were executed on a virtual machine equipped with Ubuntu 22.04 LTS, boasting 16GB of RAM, a 1TB Solid State Drive, and an Intel Core i5-5300U CPU. The software components used were of the following versions:

- Docker version 23.0.5.
- Docker-compose version 1.29.2.
- Cmake version 3.27.7.
- Python version 3.10.6 with the following modules:
  - Flask version 2.3.2.
  - matplotlib==3.7.1
  - pika==1.3.1
  - progressbar==2.5
  - docker 5.0.3
  - docker-compose 1.29.2
  - jsonschema 3.2.0

6.1 Post-processing simulation

Running the DEMO.py file creates a running flask instance that is reachable through http://localhost:5000

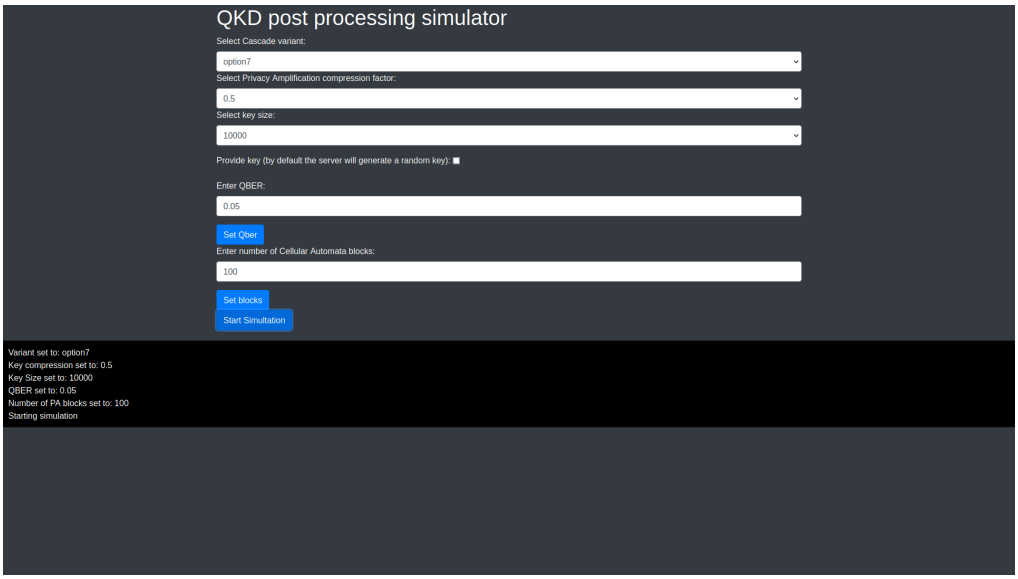


Figure 6.1. Demo web interface and setup.

After inserting all the parameter it is possible to click on the start simulation button, which will automatically create the following containers using docker Python APIs, note that for this step is important to have a docker instance running

- client docker container
- server docker container
- rabbitmq docker container

The containers will then run the simulation and output the final amplified key, as well as the corrected key.

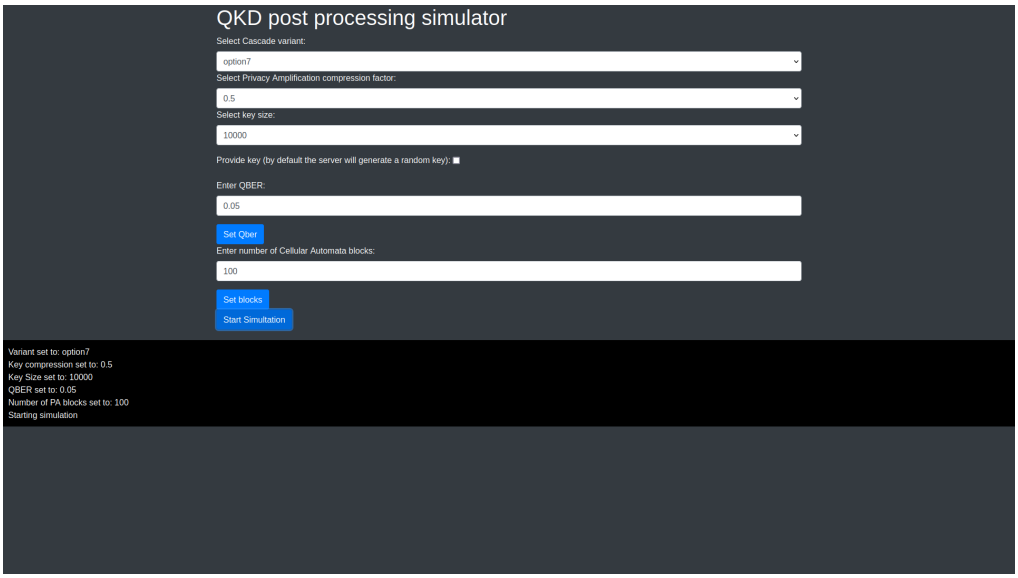


Figure 6.2. Demo web interface results.

it is also possible to retrieve the performance data from the ResultsDEMO folder, here is the resulting JSON file from the previous simulation

```
{"elapsed_real_time": 0.718586, "ask_parity_messages": 396.0, "efficiency":
    1.073335, "PA_time": 0.0247812, "Client received key":
    "01011000111...0111", "Client correct key": "010110001111...100000111",
    "Client amplified key": " 0101001111...100000100", "Remaining errors": 0}
```


Figure 6.3. JSON structure and results.

6.2 Post-processing parameters tests

The RUN_TEST.py file is used to run tests on different parameters of the EC and PA protocols. The script automatically runs the given amount of tries for a selected independent variable, keysize or QBER. The modifiable parameters are:

- QBER (when selected as independent variable it can be inserted as a range)
- keysize (when selected as independent variable it can be inserted as a range)
- number of total runs to cover the independent variable range
- PA blocks
- PA compression
- Cascade variant

The following graphs have been generated by running with QBER as the independent variable, and the followiing parameters:

- QBER form 0.0 to 0.1
- keysize= 10000
- number of total runs=100
- PA compression=0.1
- Cascade variant=all

when running tests in this way, the software will automatically generate the required containers in series with each different parameters combination exploiting the docker-compose environment file to pass parameters for the simulation to the containers.

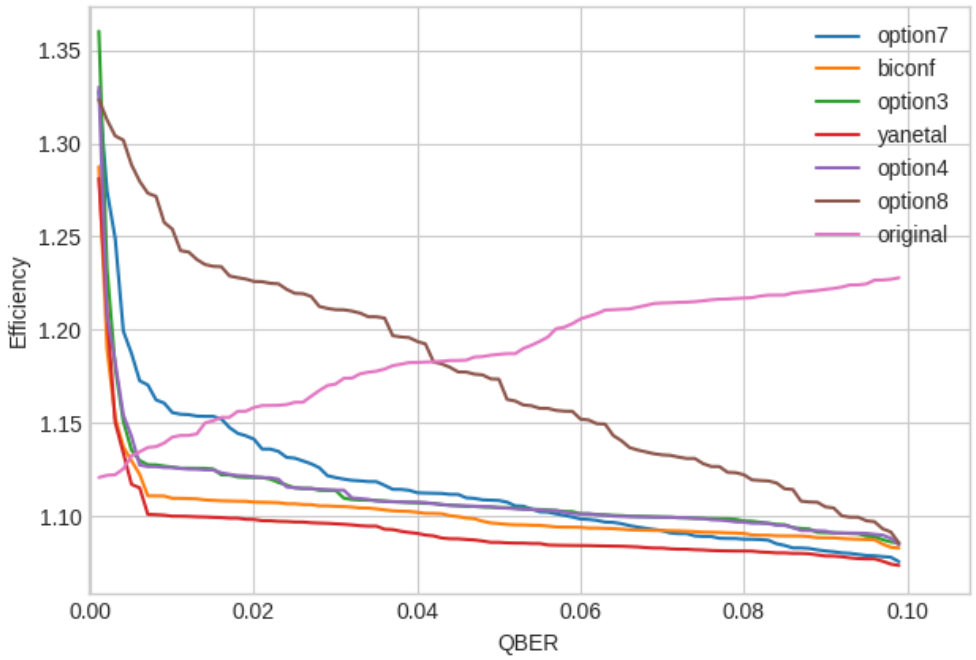


Figure 6.4. Efficiency of different Cascade version over 100 rounds.

A total of 800 different keys were processed, all the results are stored in a Results folder as JSONs. The graph above shows that the system is responsive to the change of parameters for post-processing and also that the results are consistent with those of other studies.

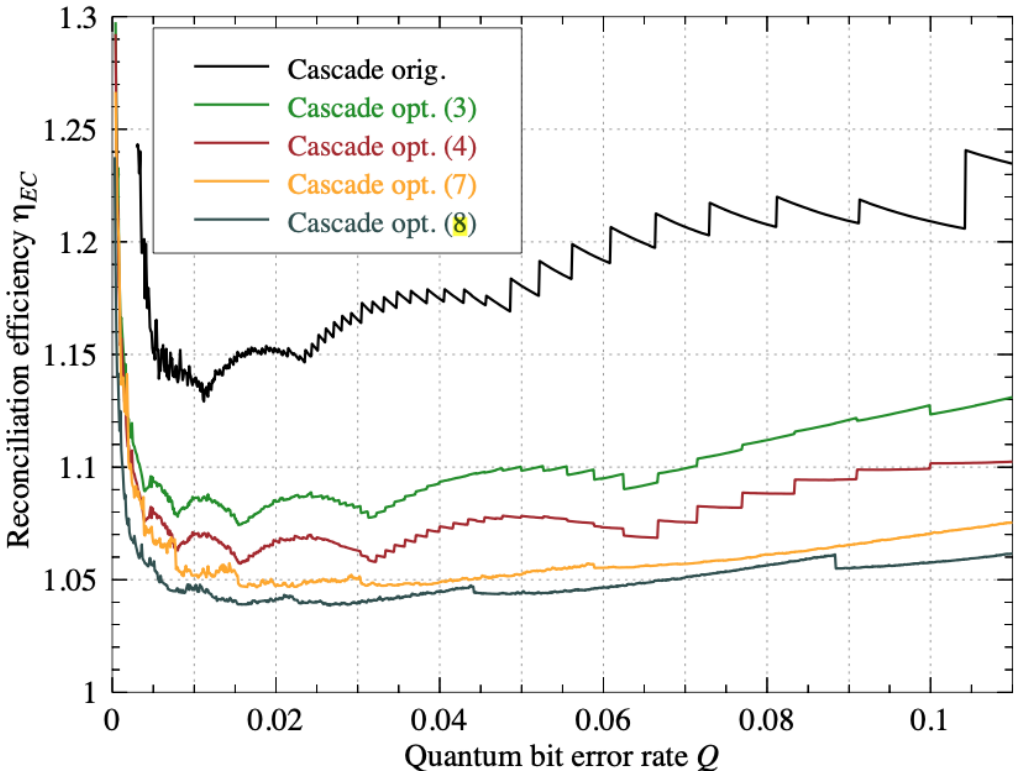


Figure 6.5. Efficiency study from [31]

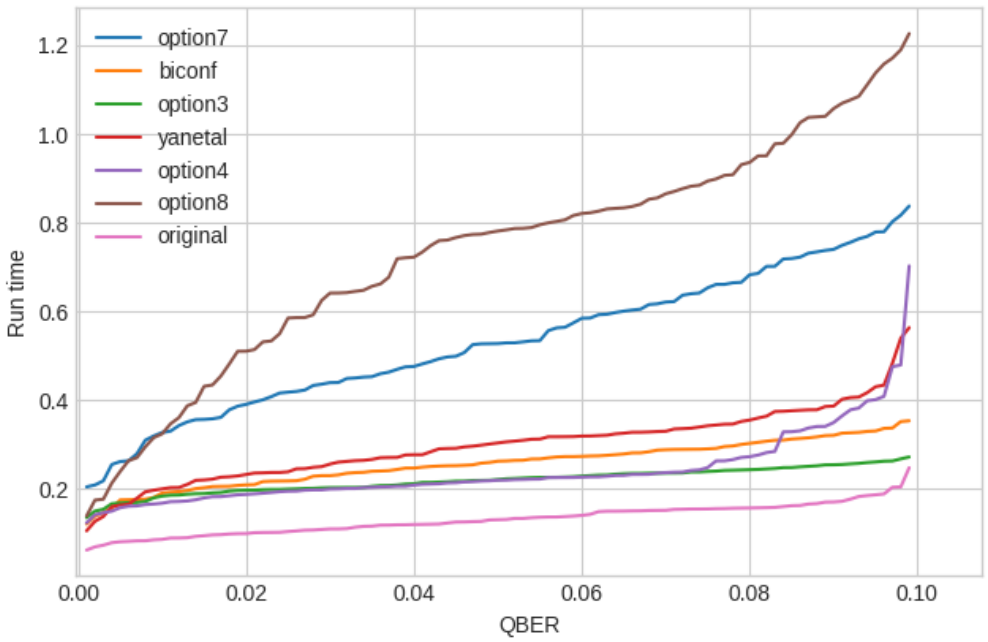


Figure 6.6. Run time of different Cascade version over 100 rounds.

of course with a greater number of runs, one can obtain a smoother curve. For instance in this test where run 10000 key siftings.

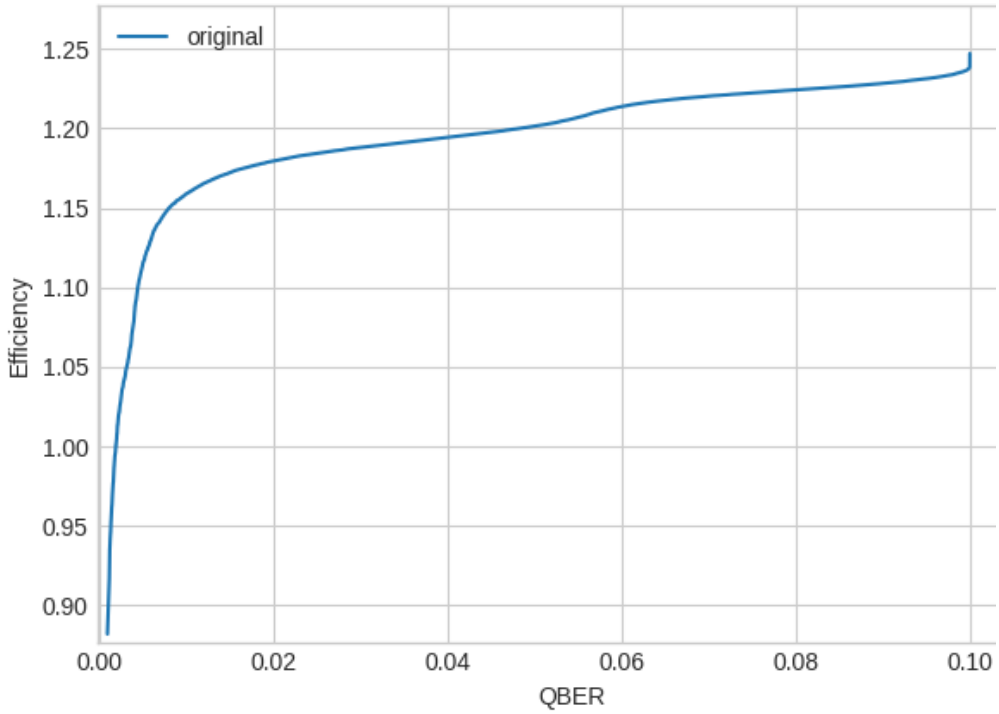


Figure 6.7. Efficiency of Cascade original over different QBERs for 10000 runs.

Chapter 7

Conclusions and future work

The aim of this work was to offer a fresh perspective on Quantum Key Distribution (QKD) post-processing. While ongoing research is dedicated to finding ways to enhance QKD post-processing with new technologies and protocols, few studies explore how to integrate these systems into everyday environments like distributed networks. Developing a comprehensive post-processing module for a QKD system is a pivotal step in QKD development. However, this work identified some challenges and proposed improvements to the overall system. The proposed implementation was developed in a lower-level language for straightforward implementation on basic hardware. Despite encountering some issues in the implementation, the outcome is a robust architecture that can be easily adopted and modified as needed.

The necessity to manage memory and performance at a lower level became critical when aiming to improve the efficiency and reliability of the system. The use of low-level programming became apparent during development and resulted in the implementation proposed in chapter 5. This specific aspect proved to be beneficial not only for the key server implementation but also for other research related to QKD post-processing. This is noteworthy as current implementations of QKD post-processing simulators often focus on exchanges within the same machine, with limited consideration for the overall system concerning individual protocols.

Tests conducted on the QKD simulator reveal a flexible architecture with some challenges in integration with softwarized networks. Optimizations in QKD post-processing protocol implementations can enhance required timings, and using TCP sockets instead of message brokers may improve overall performance. Other potential developments for this module could involve the incorporation of hardware acceleration techniques, which were not explored in this stage.

Concerning the overall architecture, the performed tests indicate that, with a tailored choice of protocol, the post-processing phase can be efficiently executed without creating a bottleneck for the entire QKD process. The proposed architecture is versatile, serving as both a testing platform and a post-processing module in QKD applications. The results demonstrate the system's adaptability to different parameters and key sifting requirements.

Future work in this area may involve employing real QKD devices to assess the flexibility of the proposed implementation when devices from different producers are used.

Bibliography

[1] F.Arute, K. Arya, and R.Babbush, "Quantum supremacy using a programmable superconducting processor", Nature, vol. 574, 2019, pp. 505–510, DOI 10.1038/s41586-019-1666-5

[2] A. G. White, D. F. V. James, W. J. Munro, and P. G. Kwiat, "Exploring hilbert space: Accurate characterization of quantum information", Physical review. A, Atomic, molecular, and optical physics, vol. 65, no. 1, 2002, DOI 10.1103/PhysRevA.65.012301

[3] P. A. M. Dirac, "A new notation for quantum mechanics", Mathematical Proceedings of the Cambridge Philosophical Society, vol. 35, no. 3, 1939, pp. 416–418, DOI 10.1017/S0305004100021162

[4] M. Benslama, "Quantum communications in new telecommunications systems", ISTE Ltd/John Wiley and Sons Inc, 2008, ISBN: 1-119-33251-6

[5] M. Born, "Quantum mechanics of collision processes", Zeit. Physik, vol. 37, Jun 1926, pp. 863–867, DOI 10.1007/BF01397477

[6] C. H. Bennett, G. Brassard, and N. D. Mermin, "Quantum cryptography without bell's theorem", Phys. Rev. Lett., vol. 68, Feb 1992, pp. 557–559, DOI 10.1103/PhysRevLett.68.557

[7] M. N. Wegman and J. Carter, "New hash functions and their use in authentication and set equality", Journal of computer and system sciences, vol. 22, Nov 1981, pp. 265–279, DOI https://doi.org/10.1016/0022-0000(81)90033-7

[8] A. Carrasco-Casado, V. Fernández, and N. Denisenko, "Free-Space Quantum Key Distribution", Optical Wireless Communications (M. Uysal, C. Capsoni, Z. Ghassemlooy, A. Boucouvalas, and E. Udvary, eds.), pp. 589–607, Springer International Publishing, 2016, DOI 10.1007/978-3-319-30201-0_27

[9] C. H. Bennett, D. Zekrifa, and J. Robert, "Privacy amplification by public discussion", SIAM J. Comput., vol. 17, 2012, pp. 210–229, DOI 10.1137/S0097539792241371

[10] A. K. Ekert, "Quantum cryptography based on bell's theorem", Phys. Rev. Lett., vol. 67, Aug 1991, pp. 661–663, DOI 10.1103/PhysRevLett.67.661

[11] M. Epping, H. Kampermann, and D. Bruss, "Designing bell inequalities from a tsirelson bound", Phys. Rev. Lett., vol. 111, Dec 2013, DOI 10.1103/PhysRevLett.111.240404

[12] N. Alshaer, A. Moawad, and A. Ismail, "Reliability and security analysis of an entanglement-based qkd protocol in a dynamic ground-to-uav fso communications system", IEEE Access, vol. PP, Dec 2021, pp. 1–1, DOI 10.1109/ACCESS.2021.3137357

[13] U. Vazirani and T. Vidick, "Fully device independent quantum key distribution", Commun. ACM, vol. 62, Mar 2019, p. 133, DOI 10.1145/3310974

[14] H. P. Yuen, "Security of Quantum Key Distribution", IEEE Access, vol. 4, Feb 2016, pp. 724–749, DOI 10.1109/ACCESS.2016.2528227

[15] Quantiki, `https://www.quantiki.org/wiki/bb84-and-ekert91-protocols`

[16] P. W. Shor and J. Preskill, "Simple Proof of Security of the BB84 Quantum Key Distribution Protocol", Physical Review Letters, vol. 85, Jul 2000, pp. 441–444, DOI 10.1103/PhysRevLett.85.441

[17] C. Portmann and R. Renner, "Security in Quantum Cryptography", Reviews of Modern Physics, vol. 94, Jun 2022, p. 025008, DOI 10.1103/RevModPhys.94.025008

[18] R. Renner and S. Wolf, "Simple and Tight Bounds for Information Reconciliation and Privacy Amplification", Advances in Cryptology - ASIACRYPT 2005 (D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, and B. Roy, eds.), pp. 199–216, Springer Berlin Heidelberg, 2005, DOI 10.1007/11593447_11

[19] C. Bennett, G. Brassard, C. Crepeau, and U. Maurer, "Generalized privacy amplification", IEEE Trans. Inf. Theory, vol. 41, 1995, pp. 1915–1923, DOI 10.1109/18.476316

[20] Y. Lu, E. Bai, X. qin Jiang, and Y. Wu, "High-Speed Privacy Amplification Algorithm Using Cellular Automate in Quantum Key Distribution", Electronics, vol. 11, Aug 2022, DOI 10.3390/electronics11152426

[21] H. Yan, T. Ren, X. Peng, X. Lin, W. Jiang, T. Liu, and H. Guo, "Information Reconciliation Protocol in Quantum Key Distribution System", 2008 Fourth International Conference on Natural Computation, Jinan, Shandong, China, 2008, pp. 637–641, DOI 10.1109/ICNC.2008.755

[22] B. Rijsman, https://github.com/brunorijsman/cascade-python

[23] C. H. Bennett, F. Bessette, G. Brassard, L. Salvail, and J. Smolin, "Experimental quantum cryptography", Journal of Cryptology, vol. 5, pp. 3–28, DOI https://doi-org.ezproxy.biblio.polito.it/10.1007/BF00191318

[24] B. Gilles and S. Louis, "Secret-Key Reconciliation by Public Discussion", Advances in Cryptology, vol. 765, 1994, pp. 410–423, DOI 10.1007/3-540-48285-7_35

[25] W. T. Buttler, S. K. Lamoreaux, J. R. Torgerson, G. H. Nickel, C. H. Donahue, and C. G. Peterson, "Fast, efficient error reconciliation for quantum cryptography", Physical Review A, vol. 67, May 2003, DOI 10.1103/PhysRevA.67.052303

[26] I. Ulidowski, I. Lanese, U. P. Schultz, and C. Ferreira, "Reversible Computation: Extending Horizons of Computing: Selected Results of the COST Action IC1405", Springer International Publishing, 2020, ISBN: 978-3-030-47360-0 978-3-030-47361-7

[27] R. Gallager, "Low-density parity-check codes", IEEE Transactions on Information Theory, vol. 8, Jan 1962, pp. 21–28, DOI 10.1109/TIT.1962.1057683

[28] D. Elkouss, J. Martinez-Mateo, and V. Martin, "Information reconciliation for quantum key distribution." arXiv:1007.1616, 2011, DOI 10.48550/arXiv.1007.1616

[29] C. Elliott, A. Colvin, D. Pearson, O. Pikalo, J. Schlafer, and H. Yeh, "Current status of the darpa quantum network." arXiv:quant-ph/0503058, 2005, DOI 10.48550/arXiv.quant-ph/0503058

[30] R. W. Hamming, "Error detecting and error correcting codes", The Bell System Technical Journal, vol. 29, Apr 1950, pp. 147–160, DOI 10.1002/j.1538-7305.1950.tb00463.x

[31] J. Martinez-Mateo, C. Pacher, M. Peev, A. Ciurana, and V. Martin, "Demystifying the Information Reconciliation Protocol Cascade." arXiv:1407.3257, December 2014, DOI https://doi.org/10.48550/arXiv.1407.3257

[32] D. Slepian and J. Wolf, "Noiseless coding of correlated information sources", IEEE Transactions on Information Theory, vol. 19, Jul 1973, pp. 471–480, DOI 10.1109/TIT.1973.1055037

[33] G. Limei, R. Qi, and J. D. andHuang Duan, "QKD Iterative Information Reconciliation Based on LDPC Codes", International Journal of Theoretical Physics, vol. 59, Jun 2020, pp. 1717–1729, DOI 10.1007/s10773-020-04438-9

[34] A. Shokrollahi, "An Introduction to Low-Density Parity-Check Codes", Theoretical Aspects of Computer Science: Advanced Lectures (G. B. Khosrovshahi, A. Shokoufandeh, and A. Shokrollahi, eds.), pp. 175–197, Springer, 2002, DOI 10.1007/3-540-45878-6_6

[35] A. Liveris, Z. Xiong, and C. Georghiades, "Compression of binary sources with side information at the decoder using LDPC codes", IEEE Communications Letters, vol. 6, Oct 2002, pp. 440–442, DOI 10.1109/LCOMM.2002.804244

[36] D. Elkouss, A. Leverrier, R. Alléaume, and J. Boutros, "Efficient reconciliation protocol for discrete-variable quantum key distribution", 2009 IEEE International Symposium on Information Theory, Paris (France), June, 2009, pp. 1879–1883, DOI 10.1109/ISIT.2009.5205475

[37] R. Storn and K. Price, "Minimizing the real functions of the ICEC'96 contest by differential evolution", Proceedings of IEEE International Conference on Evolutionary Computation, Nagoya (Japan), May, 1996, pp. 842–844, DOI 10.1109/ICEC.1996.542711

[38] A. Shokrollahi and R. Storn, "Design of efficient erasure codes with differential evolution", 2000 IEEE International Symposium on Information Theory (Cat. No.00CH37060), Sorrento (Italy), Jun 25-30, 2000, pp. 5–, DOI 10.1109/ISIT.2000.866295

[39] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes", IEEE Transactions on Information Theory, vol. 47, Feb 2001, pp. 619–637, DOI 10.1109/18.910578

[40] V. Scarani and R. Renner, "Quantum cryptography with finite resources: unconditional security bound for discrete-variable protocols with one-way post-processing", Physical Review Letters, vol. 100, May 2008, DOI 10.1103/PhysRevLett.100.200501

[41] R. Wolf, "Quantum key distribution: An introduction with exercises", Springer, 2021, ISBN: 978-3-030-73991-1

[42] N. Gisin, G. Ribordy, W. Tittel, and H. Zbinden, "Quantum cryptography", Rev. Mod. Phys., vol. 74, Mar 2002, pp. 145–195, DOI 10.1103/RevModPhys.74.145

[43] C. Weedbrook, S. Pirandola, R. García-Patrón, N. Cerf, T. Ralph, J. Shapiro, and S. Lloyd, "Gaussian quantum information", Rev. Mod. Phys., vol. 84, no. 2, 2012, pp. 621–669, DOI 10.1103/RevModPhys.84.621

[44] F. Grosshans and P. Grangier, "Continuous variable quantum cryptography using coherent states", Phys. Rev. Lett., vol. 88, no. 5, 2002, DOI 10.1103/PhysRevLett.88.057902

[45] G. Gilbert and M. Hamrick, "Secrecy, computational loads and rates in practical quantum cryptography", Algorithmica, vol. 34, 2002, pp. 314–339, DOI 10.1007/s00453-002-0983-y

[46] R. Melki, H. Noura, M. Mansour, and A. Chehab, "A survey on ofdm physical layer security", Phys. Commun., vol. 32, 2019, pp. 1–30, DOI 10.1016/j.phycom.2018.10.006

[47] M. Bottarelli, G. Epiphaniou, D. B. Ismail, P. Karadimas, and H. Al-Khateeb, "Physical characteristics of wireless communication channels for secret key establishment: A survey of the research", Comput. Secur., vol. 78, 2018, pp. 454–476, DOI 10.1016/j.cose.2018.07.005

[48] J. Zhang, T. Duong, A. Marshall, and R. Woods, "Key generation from wireless channels: A review", IEEE Access, vol. 4, 2017, pp. 614–626, DOI 10.1109/ACCESS.2016.2637003

[49] J. Carter and M. Wegman, "Universal classes of hash functions", J. Comput. Syst. Sci., vol. 18, 1979, pp. 143–154, DOI 10.1016/0022-0000(79)90044-8

[50] B. Tang, B. Liu, Y. Zhai, C. Wu, and W. Yu, "High-speed and large-scale privacy amplification scheme for quantum key distribution", Sci. Rep., vol. 9, 2019, DOI 10.1038/s41598-019-51926-0

[51] X. Wang, Y. Zhang, S. Yu, and H. Guo, "High-speed implementation of length-compatible privacy amplification in continuous-variable quantum key distribution", IEEE Photonics J., vol. 10, 2018, DOI 10.1109/JPHOT.2018.2865305

[52] M. Zidan, S. Aldulaimi, and H. Eleuch, "Analysis of the quantum algorithm based on entanglement measure for classifying boolean multivariate function into novel hidden classes: Revisited", Appl. Math. Inf. Sci., vol. 15, 2021, pp. 643–647, DOI 10.18576/amis/150149

[53] S.-S. Yang, Z.-L. Bai, X.-Y. Wang, and Y.-M. Li, "Fpga-based implementation of size-adaptive privacy amplification in quantum key distribution", IEEE Photonics J., vol. 9, 2017, DOI 10.1109/JPHOT.2017.2760659

[54] D.-W. Li, P. Huang, Y.-M. Zhou, Y. Li, and G.-H. Zeng, "Memory-saving implementation of high-speed privacy amplification algorithm for continuous-variable quantum key distribution", IEEE Photonics J., vol. 10, 2018, DOI 10.1109/JPHOT.2018.2869859

[55] E.-J. Bai, X.-Q. Jiang, and Y. Wu, "Memory-saving and high-speed privacy amplification algorithm using lfsr-based hash function for key generation", Electronics, vol. 11, 2022, p. 377, DOI 10.3390/electronics11030377

[56] L. Bassham, A. Rukhin, J. Soto, J. Nechvatal, M. Smid, S. Leigh, M. Levenson, M. Vangel, N. Heckert, and D. Banks, "A statistical test suite for random and pseudorandom number generators for cryptographic applications." NIST SP800-22, 2001

[57] A. Obada, D. Abo-Kahla, N. Metwally, and M. Abdel-Aty, "The quantum computational speed of a single cooper pair box", Phys. E Low-Dimens. Syst. Nanostruct., vol. 43, 2011, pp. 1792–1797, DOI 10.1016/j.physe.2011.05.009

[58] M. Zidan, A. Abdel-Aty, A. Khalil, M. Abdel-Aty, and H. Eleuch, "A novel efficient quantum random access memory", IEEE Access, vol. 9, 2021, pp. 151775–151780, DOI 10.1109/ACCESS.2021.3084556

[59] C. H. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing", Theor. Comput. Sci., vol. 560, 2014, pp. 7–11, DOI 10.1016/j.tcs.2014.09.004

[60] E. Diamanti and A. Leverrier, "Distributing secret keys with quantum continuous variables: Principle, security and implementations", Entropy, vol. 17, 2015, pp. 6072–6092, DOI 10.3390/e17096072

[61] B.-Y. Tang, B. Liu, Y.-P. Zhai, C.-Q. Wu, and W.-R. Yu, "High-speed and Large-scale Privacy Amplification Scheme for Quantum Key Distribution", Scientific Reports, vol. 9,

Oct 2019, DOI 10.1038/s41598-019-50290-1
[62] M. Hayashi and T. Tsurumaru, "More efficient privacy amplification with less random seeds via dual universal hash function", IEEE Transactions on Information Theory, vol. 62, 2016, pp. 2213–2232, DOI 10.1109/TIT.2016.2526018
[63] B. Liu, B.-K. Zhao, W.-R. Yu, and C.-Q. Wu, "Fit-pa: Fixed scale fft-based privacy amplification algorithm for quantum key distribution", Journal of Internet Technology, vol. 17, 2016, pp. 309–320, DOI 10.6138/JIT.2016.17.2.20150703e
[64] M. Hayashi, "Exponential decreasing rate of leaked information in universal random privacy amplification", IEEE Transactions on Information Theory, vol. 57, no. 6, 2011, pp. 3989–4001, DOI 10.1109/TIT.2011.2110950
[65] X. Ma, C.-H. Fung, and H.-K. Lo, "Quantum key distribution with entangled photon sources", Physical Review A, vol. 76, Jul 2007, DOI 10.48550/arXiv.quant-ph/0703122
[66] D. Li, P. Huang, Y. Zhou, Y. Li, and G. Zeng, "Memory-Saving Implementation of High-Speed Privacy Amplification Algorithm for Continuous-Variable Quantum Key Distribution", IEEE Photonics Journal, vol. 10, Oct 2018, pp. 1–12, DOI 10.1109/JPHOT.2018.2865486
[67] M. N. Wegman and J. L. Carter, "New hash functions and their use in authentication and set equality", Journal of computer and system sciences, vol. 22, no. 3, 1981, pp. 265–279, DOI https://doi.org/10.1016/0022-0000(81)90033-7
[68] N. W. et al., "A fast and versatile qkd system with hardware key distillation and wavelength multiplexing", New Journal of Physics, vol. 16, Jan 2013, p. 013047, DOI https://doi.org/10.1088/1367-2630/16/1/013047
[69] G. van Assche, "Quantum cryptography and secret-key distillation", Cambridge University Press, 2006, ISBN: 9780511617744
[70] Z. C. et al., "Fast implementation of length-adaptive privacy amplification in quantum key distribution", Chinese Physics B, vol. 23, Sep 2014, DOI 10.1088/1674-1056/23/9/090310
[71] M. Ben-Or, "Probabilistic algorithms in finite fields", SIAM Journal on Computing, vol. 9, no. 2, 1980, pp. 273–280, DOI 10.1137/0209024
[72] H. Krawczyk, "Lfsr-based hashing and authentication", Advances in Cryptology, Berlin (Germany), 1994, pp. 129–139, DOI 10.1007/3-540-48658-5_15
[73] S. Wolfram, "Statistical mechanics of cellular automata", Rev. Mod. Phys., vol. 55, Jul-Seè 1983, pp. 601–644, DOI 10.1103/RevModPhys.55.601
[74] S. Wolfram, "Universality and complexity in cellular automata", Phys. D Nonlinear Phenom., vol. 10, no. 1, 1984, pp. 1–35, DOI 10.1016/0167-2789(84)90245-8
[75] S. E.Yacoubi, "A mathematical method for control problems on cellular automata models", Int. J. Syst. Sci., vol. 39, May 2008, pp. 529–538, DOI 10.1080/00207720802191339
[76] D. A. Rosenblueth and C. Gershenson, "A model of city traffic based on elementary cellular automata", Complex Syst., vol. 19, Jan 2011, pp. 305–322, DOI 10.25088/ComplexSystems.19.4.305
[77] B. Teklu, A. Ferraro, M. Paternostro, and M. G. A. Paris, "Nonlinearity and nonclassicality in a nanomechanical resonator", EPJ Quantum Technol., vol. 2, Jan 2015, DOI 10.1140/epjqt/s40507-015-0027-1
[78] A. Menezes, P. C. Oorschot, and S. Vanstone, "Handbook of applied cryptography", CRC Press, Oct 1997, ISBN: 0-8493-8523-7
[79] M. Luby, "Pseudorandomness and cryptographic applications", Princeton University Press, Jan 1996, ISBN: 978-0-691-02546-9
[80] M. Tomassini, M. Sipper, and M. Perrenoud, "On the generation of high-quality random numbers by two-dimensional cellular automata", IEEE Trans. Comput., vol. 49, Nov 2000, pp. 1146–1151, DOI 10.1109/12.868675
[81] C. M. Software, `hhttps://github.com/CopernicaMarketingSoftware/AMQP-CPP`

Appendix A

User manual

This chapter breaks down the steps to get the proposed project up and running, digging into the main choices made for how it works. The Error Correction sub module and the Privacy Amplification one are built into a single container through a Docker Compose file.

A.1 QKD Post-processing simulator

The simulator can be used in two main modes, creating a dockerized environment or running it inside a Linux host machine (virtual or physical), on the local network.

A.1.1 Running the simulation in a dockerized environment

The simulator is composed of three containers, a client, a server one and a Rabbitmq one. They can be run by executing a docker compose file with:
`docker-compose build`

Docker compose used is version 3.8. The configuration file will build the client, server and Rabbitmq images.

Once the images are built, they can be used to run experiments on post-processing. The experiments can be run in different ways that will be now explained

Running the RUN_TESTS script

The RUN_TEST can be automatically run using the provided python API, the interface is the python method runMultipleTests() and takes the following arguments:

1. **independent_variable** that can be either the QBER or the key size and is passed as a string value
2. **variant** that can be a list of those present in table 5.1, the argument is passed as a string value with the names of the variants separated by commas, the string 'all' will execute the experiment for all variants (i.e. the total number of runs will be $runs \times \#variants$
3. **compression** float value of PA compression ratio, possible values are 0.1, 0.2, 0.4 and 0.5
4. **qber** float value of QBER, up to 6 digits of precision, if the QBER is also the independent variable, this argument must be a length-two list where the first element is the starting QBER and the last the final one
5. **keysize** integer value of key size, if the key size is also the independent variable, this argument must be a length-two list where the first element is the starting QBER and the last the final one

6. **runs** number of key sifting to execute while proceeding from the starting value of the independent variable to the final one

Example: `runMultipleExperimets('qber', 'all', 0.5, [0.01, 0.05], 10000, 100)` Running this python code would make the script run 700 ($runs \times n_{variants} = 100 \times 7$) tests of key post-processing

With this python method one can implement multiple simulation patterns and test different conditions.

The RUN_TESTS script can also be run by command line just by using python3
`python3 RUN_TESTS.py [independent variable] [variant] [compression] [qber/keysize] [keysize/qber] [paBlocks] [port] [man_port] [runs]`

Running the DEMO script

The DEMO script is intended to provide an easy method to perform customized single-run tests. When running the DEMO script a web app is avaiable on http://localhost:5000 and provides a easy interface to set the privacy amplification parameters and run a simulation.

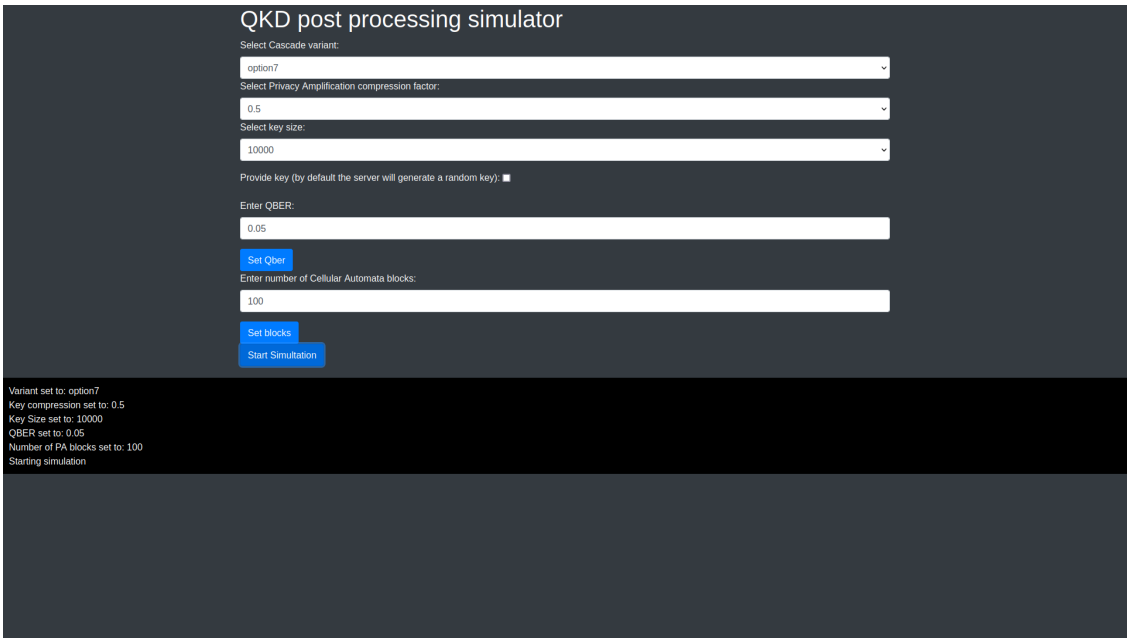


Figure A.1. Demo web interface and setup.

When running a simulation in this way, the docker containers are automatically built and start running the simulation. The simulation result is then showed in the web interface and a more detailed description is saved in a json file inside the ResultsDEMO directory. The DEMO script can be run from command line just by inserting
`python3 DEMO.py`

Running via docker compose

One can also run a simulation just by inserting `docker-compose up`. In this case the containers will be built and will start the simulation by finding the parameters inside the environment(.env) file and the file must contain the following variables

- **TAG:** the environment version used.

- **PORT:** the TCP port used by Rabbitmq message service.
- **MAN_PORT:** the TCP port used to connect to Rabbitmq management service.
- **USR:** username used to connect to Rabbitmq.
- **PW:** password used to connect to Rabbitmq.
- **QBER:** test QBER to run an experiment.
- **NBITS:** test key length to run an experiment.
- **ER_VARIANT:** cascade configuration to run an experiment.
- **COMPRESSION:** privacy amplification compression ratioto run an experiment.
- **NBLOCKS:** number of cellular automata T blocks to run an experiment.
- **RUNS:** number of key to post-process to run the experiments.
- **HOST:** name of rabbitmq host name.
- **SEQ:** each experiments has a sequential number.
- **RANDOM_KEY:** option to run the experiment using random key, if set to false, the following variable must be KEY.

Please note that when running a simulation in this way, no real result is provided to the user, this method is intended only to test if the system can be built and run correctly.

A.1.2 Running the simulation on a Linux host

To run the simulation between two Linux hosts is necessary to install the required components first and it can be done by running the install.sh script. Note that when running the simulation in this way, the system will still need a running Rabbitmq instance, this can be in either host or in a third one.

```
#!/bin/bash

# Update package list
sudo apt-get update

# Install necessary dependencies
sudo apt-get install -y build-essential wget libboost-dev libboost-system-dev
    libboost-thread-dev libevent-dev librabbitmq-dev git python3 python3-pip
    libssl-dev snapd

sudo snap install cmake --classic

# Clone AMQP-CPP repository and build
git clone https://github.com/CopernicaMarketingSoftware/AMQP-CPP.git
cd AMQP-CPP
mkdir build
cd build
cmake .. -DAMQP-CPP_BUILD_SHARED=ON -DAMQP-CPP_LINUX_TCP=ON
cmake --build . --target install

# Install Python dependencies
pip3 install --no-cache-dir -r requirements.txt

# Run ldconfig
sudo ldconfig

# Build error_correction and privacy_amplification
cd ErrorCorrection/Build
cmake ..
make
cd ..
cd ..
cd PrivacyAmplification/build
cmake ..
make
```

Figure A.2. install script.

Once the install.sh script was run on all hosts, the simulation can be run using the provided python API, the interface is the python method runPostProcessing() and takes the following arguments:

1. **endpoint** to set this host as either server or client, string value 'client' or 'server'
2. **keysize** integer value of key size
3. **variant** string value containing the name of one of the EC variants proposed in table 5.1
4. **qber** float value of QBER, up to 6 digits of precision
5. **host** string value of host name or address
6. **port** int value, TCP port on which Rabbitmq messaging interface is running
7. **user** Rabbimtq user
8. **password** Rabbitmq password

9. **seq** sequential number for experiment
10. **compression** float value of PA compression ratio, possible values are 0.1, 0.2, 0.4 and 0.5
11. **pa_blocks** int value, number of cellular automata T blocks
12. **runs** number of key sifting to execute
13. **randomKey** boolean value option to run the experiment using random key
14. **key** provided key if the randomKey argument is set to False
15. **ER_path** which is the path in where the Error Correction submodule was built, by default it will be ./error_correction/Mycascade
16. **PA_path** which is the path in where the Privacy Amplification submodule was built, by default it will be ./privacy_amplification/PrivacyAmplificationCPP

Example
```
runPostProcessing( 'client', 1000, 'original', 0.05,'localhost',5672,'QKD',
'QKD', 0, 0.5, 100, 1, True, './error_correction/Mycascade', './privacy_amplification/Privacy
)
```

The simulation script can also be run by command line just by using python3
```
python3 wrapper.py [ER_path], [PA_path], [endpoint], [keysize], [variant], [qber],
[host], [port], [user], [pw], [seq], [compression], [pa_blocks], [runs], [randomKey],
[key]
```

Appendix B

Developer manual

In this chapter the main implementation choices will be discussed, outlining the main libraries that led to the proposed implementation.

B.1 Rabbitmq

The use of RabbitMQ entails a series of considerations essential in the development of clients and servers. It's important to note that there isn't an official RabbitMQ library for C++; rather, there are only open-source solutions supported by the community. Consequently, in the software development, two main libraries were primarily used: Pika for the Python part and AMQP-CPP for the C++ part.

B.1.1 The AMQP-CPP library

The AMQP-CPP library offers a layered architecture that grants users the choice to handle the network layer themselves. If desired, users can manage network connections independently, creating socket connections and defining an interface for AMQP-CPP to utilize for IO operations. However, this interception of the network layer is optional. AMQP-CPP provides predefined TCP and TLS modules for those who prefer the library to manage network and TLS handling. Its flexible architecture ensures portability, independence from OS-specific IO calls, and seamless integration into various event loops. While adaptable to unconventional communication layers, it effortlessly sets up with standard TCP connections. This fully asynchronous library, ideal for high-performance applications, avoids blocking system calls and operates without the need for threads. AMQP-CPP's reliance on C++17 features mandates an up-to-date compiler supporting C++17 for usage. AMQP-CPP operates in a network-agnostic fashion. It does not do IO by itself. An object must be provided that defines the IO operations. This object is the connectionHandler and in this implementation i wrote one that could also be compatible with the software running on docker containers. By using the event library every interaction with the channel is done inside an event loop and broken upon needs. The connectionHandler header is present both in the Error Correction and Privacy Amplification submodules, it is in some sense a defining part of all the implementation.

```
#ifndef MYCASCADE_CONN_HANDLER_H
#define MYCASCADE_CONN_HANDLER_H

#include <functional>
#include <unistd.h>
#include <event2/event.h>
#include <amqpcpp/libevent.h>
#include <iostream>
#include <memory>
class MyLibEventHandler : public AMQP::LibEventHandler {
public:
    MyLibEventHandler(struct event_base *evbase) : LibEventHandler(evbase){}
    uint16_t onNegotiate(AMQP::TcpConnection *connection, uint16_t interval)
        override {
        //we don't want to use heartbeats, so we return 0
        if (interval != 0) interval = 0;
        return interval;
    }
};
class ConnHandler {
public:
    ConnHandler() {
        evbase_ = event_base_new_with_config(cfg);
        evhandler_ = new MyLibEventHandler(evbase_);
    }
    ~ConnHandler() {
        event_base_free(evbase_);
        delete evhandler_;
    }

    void Start() {
        event_base_dispatch(evbase_);
    }

    void Stop() {
        event_base_loopbreak(evbase_);
    }

    operator AMQP::TcpHandler* () {
        return evhandler_;
    }

private:
    struct event_config *cfg = event_config_new();
    event_base* evbase_;
    MyLibEventHandler *evhandler_;
};


#endif //MYCASCADE_CONN_HANDLER_H
```

Figure B.1. Connection handler header.

B.1.2 Connection check and sync

Since the simulator heavily relies on a running Rabbitmq instance, it is crucial to check for a running one upon starting the simulation. This is done at the start of every simulation by the wrapper script, in this moment, the server message queue is created to permit the client to initiate the key sifting. Generally speaking the client is the one requesting the error correction and also the privacy amplification from later on.

After the client publishes the first message it waits for a server response, this of course could lead in an unlimited wait in case the server doesn't respond, but it also syncs the two peers exchanges.

B.1.3 Running instances of Rabbitmq

To run the simulator on a network where there is a running instance of Rabbitmq it is necessary to remove the Rabbitmq docker , or to run the second instance on a different port, to stop the simulator from running a new instance of rabbitmq, the docker image must be remove from the docker compose file. Note that also the `depends_on: rabbitmq:` line must be remove borh from the server and the client

```
version: "3.8"
services:
  rabbitmq:
    image: rabbitmq:3-management
    environment:
      RABBITMQ_DEFAULT_USER: ${USR}
      RABBITMQ_DEFAULT_PASS: ${PW}
    ports:
         - "${PORT}:${PORT}"
         - "${MAN_PORT}:${MAN_PORT}"

  server:
    build:
      context: .
      dockerfile: server_dockerfile
    depends_on:
      rabbitmq:
        condition: service_started
    command: [ "python3","-u", "wrapper.py", "server"
        ,"${NBITS}","${ER_VARIANT}","${QBER}","${PORT}","${USR}","${PW}",
        "${COMPRESSION}","${NBLOCKS}","${RUNS}", "${SEQ}",
        "${HOST}","${RANDOM_KEY}", "${KEY}"]

  client:
    build:
      context: .
      dockerfile: client_dockerfile
    depends_on:
      rabbitmq:
        condition: service_started
    command: [ "python3","-u", "wrapper.py", "client"
        ,"${NBITS}","${ER_VARIANT}","${QBER}","${PORT}","${USR}","${PW}","${COMPRESSION}","$
        "${SEQ}", "${HOST}","${RANDOM_KEY}", "${KEY}"]
```

Figure B.2. Simulator docker compose file.

The alternative is to run the simulator instance on a different port, this can be done by changing it in the environment file or providing it a different port to the APIs

A last note on Rabbitmq settings is about the reserved space for queue buffers, some experiments may required hundred of thousands if not millions of messages and queues, so it is important to reserve the adequate amount of memory to it.

B.2 Error Correction submodule

The error correction submodule was developed starting from [22] and it implements a complex structure for key shuffles handling and parity requests.

B.2.1 Key handling

As explained in chapter 5, the memory management is a crucial part of the implementation. The key is saved in a list of uint64_t words and the shuffles are saved as an array of indexes. Key shuffles are also saved in cache to be reused in sequent iterations, to compute known parities and reduce channel uses

B.2.2 Message format

In this section we will explain how the message exchanged are formatted.

Initialization

This is the first exchange between the two peers and it sent by the client to start the connection. The server responds by sending the noisy key.

Header	**Content**	**Description**
messageType	'initialization'	client connects to server sending an empty initialization message
replyTo	init queue name	message queue to submit the response
Body	**Empty**	Empty body

Table B.1. Initialization message request

Header	**Content**	**Description**
messageType	'initializationReponse'	empty initialization message
nrBits	number of bits of key	number of bits inside the body of the message i.e.noisy key bits
nrWords	number of words of key	number of 64 bit words of the noisy key
Body	**Noisy key**	Server sends the noisy key

Table B.2. Initialization message response

Start iteration

In the start iteration message, the client send a message containing the key shuffle

Header	Content	Description
messageType	‘startIteration‘	client starting a new cascade iteration
seed	key shuffle	the order of bits to use in this iteration
nIteration	sequential integer	the number of the current iteration
Body	**Empty**	Empty body

Table B.3. Start iteration message

there is no start iteration response, the message is ackee by the server upon delivery.

Parities request

In order to reduce the number of classical channel uses, the parity bits request are sent in blocks, meaning that a single message is sent for all the required parities at each round. The request message is formatted as follows

Header	Content	Description
messageType	‘cascade_rpc‘	Remote Procedure Call for parities
numberOfBlocks	integer number	number of key blocks to calculate
replyTo	client queue name	message queue to submit the response
Body	**List of block indexes**	comma separated values of start and end indexes

Table B.4. Parities request

Header	Content	Description
messageType	‘cascade_rpc‘	Remote Procedure Call for parities
numberOfBlocks	integer number	number of key blocks to calculate
replyTo	client queue name	message queue to submit the response
Body	**List of block indexes**	comma separated values of start and end indexes

Table B.5. Parities response

Close channel

Ate the end of the cascade protocol iterations, the client communicate to the server to end the connection and the server replies with the correct key, with it, the client can compute the efficiency, the frame error rate and the remaining errors.

Header	Content	Description
messageType	‘closing‘	client closing the channel
Body	**Empty**	Empty body

Table B.6. Close channel request

Header	Content	Description
messageType	‘closingConfirm‘	empty initialization message
nrBits	number of bits of key	number of bits inside the body of the message i.e. correct key bits
nrWords	number of words of key	number of 64 bit words of the correct key
Body	**Correct key**	Server sends the correct key

Table B.7. Close chanel response

B.3 Privacy amplification submodule

B.3.1 Message format

In this section we will explain how the message exchanged are formatted.

Initialization

This is the first exchange between the two peers and it sent by the client to start the connection. The client sends the Privacy Amplification parameters to the server. The server does not respond directly, the message is acked automatically by Rabbitmq upon delivery.

Header messageType	**Content** 'PaNegotiaion'	**Description** initialization message
K	integer number of k	number of k blocks to use in the cellular automata algorithm
M	integer number of m	number M of bits of each k block
N	integer number of n	number N of bits of cellular automata
Body	**CA initial value**	Client sends the CA seed

Table B.8. Initialization message from client

Close channel

At the end of the privacy amplification phase, the server sends a message to confirm the correct completion of the key sifting process.

Header	**Content**	**Description**
messageType	'closingConfirm'	server signal the end of the procedure

Table B.9. Post-process close