Quantum Cryptography and Security

Error correction

Paolo Lapo Cerni
paololapo.cern/at/studenti.unipd.it
University of Padua

6th December 2023

---

1 Introduction

The problem we want to address is the one of *quantum key distribution*, i.e. the sharing between Alice and Bob of a string of random numbers they can use as a (symmetric) cryptographic key. While most of the classical approaches are based on mathematical constraints on the attacker's computational resources, the security of a quantum protocol is given by the analysis of all the information possibly revealed to the eavesdropper (Eve) along the communications.
In practice, we can divide the problem of QKD into sequential steps:

Random sharing: transmission, manipulation, and detection of qubits to obtain some correlation between the variables X (on Alice's side) and Y (on Bob's side). This needs to be done by sharing an entangled state over a quantum channel.

Advantage distillation: filtering out the low-correlated values to obtain high correlation between X and Y, i.e. advantage over the eavesdropper. In the BB84 protocol, it is the sifting of the results of the measurements for which Alice and Bob have chosen the same basis.

Error-rate estimation: the quantum communications introduce mismatches between X and Y (which would be expected to be equal). It's crucial to estimate the magnitude of these errors because most of the protocols need this information to tune the parameters. If the error rate is higher than a chosen threshold, then Alice and Bob abort the protocol.

Information reconciliation: error correction of the mismatches. Alice and Bob need to cooperate over a classic channel to reconcile X and Y. In so doing, some bits will be leaked to Eve decreasing the privacy of the key.

Error verification: Alice and Bob compare some ε-universal hash of the keys to establish if X is actually equal to Y after error-correction.

Privacy amplification: make the key uniformly distributed, i.e. remove side information and redundancy from the keys with minimum compression. It's the same problem of extracting randomness in the context of *quantum random number generation*.

This report will mainly focus on the information reconciliation step, comparing three different protocols: LDPC hashing, Cascade, and Winnow. These different approaches will be briefly introduced, simulated via some Python code, and analyzed using two different definitions of *efficiency*.

2 Methods and analysis

2.1 Information reconciliation protocols

In this subsection, we introduce the error correction protocols we compared in this report.

Cascade

The first one is called Cascade and it's based on parity comparisons and the *binary search* for errors [1]. The underlying assumption is that the errors are evenly distributed throughout the transmitted key. Then, by dividing the keys into blocks and comparing the parity of each block through the public channel, Alice and Bob can spot the blocks with errors and correct them by iterating this procedure within the blocks, dividing each block with an error into two subblocks.
With this exponentially fast binary search, they are able, at each iteration, to spot only the blocks with an odd number of errors (otherwise the parity would be the same). To mitigate this problem, they can randomly permute the keys and repeat the procedure for blocks of increasing sizes. Spotting new errors in the shuffled keys, they gain also some information about the location of other errors, previously undetected due to the even number. The protocol is better described below.
The key parameter of Cascade is k_1, the block size of the first iteration. Some empirical analyses propose the value $k_1 = 0.73/q$ as the optimal starting point, where q is the estimated QBER [2].

Algorithm 1 Cascade

Input: X, Y, k_1
Output: X, Y such that $Y \approx X$
Initialize $k = k_1$
for a given number of iterations **do**

▷ First step

Split X and Y into blocks of length k
Compute the parity of each block
Compare the parity of each block and detect the blocks with (an odd number of) errors
Perform the binary search in those blocks and correct the errors

▷ Second step

Randomly permute X and Y: X_p, Y_p
Split X_p and Y_p into blocks of length $2k$
Compute the parity of each block
Compare the parity of each block and detect the blocks with (an odd number of) errors
Perform the binary search in those blocks and correct the errors

▷ Third step

Inverse-permute X_p and Y_p: X', Y'
Compare Y and Y' and find the blocks with mismatches
Perform again the binary search in the correspondent k-length blocks and correct errors
Update $k = 2k$
end for

At each iteration or step i, Alice and Bob are disclosing the parity bits over the public channel, decreasing the privacy of the key. In particular, they reveal 1 bit per block plus $\lceil \log_2 k_i \rceil$ bits if the block had an error. Thus, there is a trade-off between the error correction capability and

the number of bits revealed to the attacker. However, most errors are corrected in the first two iterations [3].

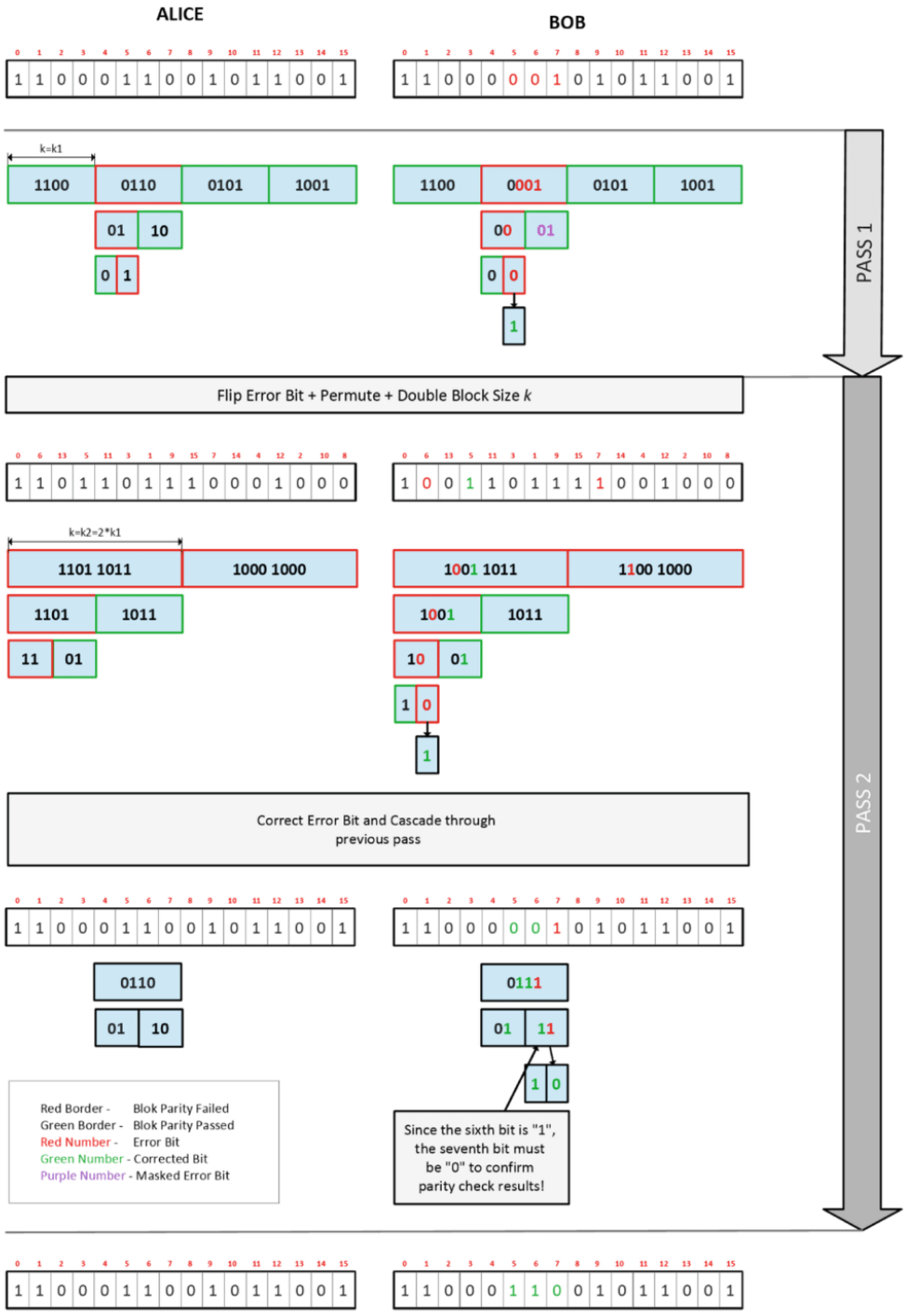

Figure 1. First two passes of a Cascade protocol. Schema from [1].

Winnow

Winnow was introduced to eliminate the binary search step in Cascade and reduce its interactivity [4]. It's based on Hamming codes and *syndrome decoding*, but with the intuition of a preliminary parity check to disclose the syndrome bits only in the case of blocks with errors.
Alice and Bob divide their keys into blocks of size $k = 2^m$ (recommended starting size is $k = 8$), compare the parities of each block, discard the last bit (the so-called *privacy maintenance*), and perform the syndrome decoding of those with (an odd number of) errors. The decoding is based on the syndromes S_A and S_B, obtained by multiplying the blocks (of length $k-1$) with a parity matrix H in the form:

$$H_{i,j}^{(m)} = \lfloor \frac{j}{2^{i-1}} \rfloor (\text{mod} 2)$$

where $i \in 1, ..., m$ and $j \in 1, ..., k-1$. If the syndrome difference $S_d = S_A \oplus S_B$ is different from $\{0\}^m$ then the error was in the first $k-1$ bits and can be corrected by the hashing procedure. The decimal representation of S_d indicates the position of the error.
Due to the reliance on Hamming codes, the Winnow protocol can correct only one bit per block and may introduce new errors.

Algorithm 2 Winnow

Input: X, Y , $k = 2^m$, H
Output: X, Y such that $Y \approx X$
Split X and Y into blocks of length k
Compute the parity of each block
Compare the parity of each block and detect the blocks with (an odd number of) errors
for each block with an error **do**
    Remove the last bit
    Compute the syndromes S_A and S_B
    Compute the syndromes difference $S_d = S_A \oplus S_B$
    if $S_d \neq \{0\}^m$ **then**
        Correct the error in the Y subblock
        Re-attach the last bit
    else
        Flip the last bit and re-attach it
    end if
end for

To sum up, Winnow discloses 1 bit for each block and m more if that block had an error.

LDPC codes

The *Low-Density Parity Check Codes* are a class of (n, k) linear codes that provides a good trade-off between algorithmic complexity, speed, and accuracy, nearly achieving the Shannon limit for the channel capacity of reliable transmission.
These codes involve a *Sparse Parity Check matrix* of size $(n-k) \times n$ that can be characterized by three parameters: n, ρ, and λ. Here, n is the coded length, ρ refers to the number of one in a row and λ to the ones in a column. The term "sparse" means that the number of ones is significantly lower than the number of zeros. Moreover, we can divide such parity matrices into two types, the

regular and *irregular* ones, depending on whether they are designed by fixed ρ and λ or probability mass functions p_ρ and p_λ.
These codes can be represented mainly in two ways: as matrices (as all the other linear block codes) or as *Tanner Graph* where the C_i check node is connected to the V_j variable node if the element $H_{i,j}$ of the correspondent H matrix is 1.
In general, optimal decoding is an NP-hard problem. However, algorithms based on iterative *belief propagation* achieve satisfactory performance and can be implemented in practice.

Typically, the error rate is an a priori unknown that needs to be estimated for each exchange. This makes designing the code after the error estimation almost unfeasible in every real scenario, leading to the so-called *rate modulation* [5]. Two common strategies to adapt the rate are *puncturing* and *shortening*.

Puncturing: removing p symbols from the codeword, i.e. converting a (n, k) code into a $(n-p, k)$ one. This procedure decreases the redundancy (and so the error correction capability) but increases the rate of transmission. It can be seen as a transmission of p bits over a binary erasure channel (BEC).

Shortening: removing s symbols during the encoding process, i.e. converting a (n, k) code into a $(n-s, k-s)$ one. This decreases the dimension of the code while keeping the redundancy, resulting in an improved correction capability but a reduced rate. It can be thought of as the assumption of the true value of s variable bits.

Given the initial code rate $R_0 = k/n$, the modulated rate is computed as:

$$R = \frac{k-s}{n-p-s}$$

2.2 Metrics of performance

In this report, we use two different metrics to quantify the efficiency of the error correction protocols. Both of them are based on the Slepian-Wolf bound, which relates the number of bits leaked during the information reconciliation (λ_{EC}) and $nH(X|Y)$ [6]. The latter can be interpreted as the minimum amount of information required to successfully reconcile the keys with an arbitrarily low failure probability in the asymptotic limit of infinite length. Under the assumptions of a qubit symmetric channel and independent and identically distributed X, the conditional entropy can be written as:

$$H(X|Y) = h_2(q)$$

where $h_2(q)$ is the binary entropy of the QBER, i.e. the symbol error rate between X and Y after the communication.
The first definition of efficiency, f_1, refers to the ratio:

$$f_1 = \frac{\lambda_{\mathrm{EC}}}{nh_2(q)}$$

It's important to notice that the optimal value is $f_1 = 1$, while $f_1 > 1$ corresponds to leaking more bits than required (due to the difficulties in designing an efficient information reconciliation protocol).

The second efficiency measure, f_2, which is probably more intuitive, evaluates the difference from the optimal value, leading to:

$$f_2 = \frac{\lambda_{\mathrm{EC}}}{n} - h_2(q)$$

3 Results

In this section, we will briefly present the results of the Python simulations. The ones referring to the LDPC codes are made by using Avesani's library available on his GitHub profile. All the results are derived from the mean of 10 independent trials.

Firstly, we compared the Cascade and Winnow protocols, using keys of length $n = 10^5$ bits. We tested the performances with different setups and hyperparameters, such as k_1 and the number of iterations for Cascade, and the $m = \log_2 k$ parameter for Winnow.
We can see that Cascade achieves better performances in terms of both efficiency and error correction capability. This can be explained by the greater adaptability of this protocol, which not only dynamically changes k_1 but also permutes the keys to gain some additional information. On the other hand, Winnow can correct only one error per block due to the reliance on Hamming codes and so it fails to reconcile each block $\underline{y}_i$ of length k with probability:

$$P_f(q) = P[d_H(\underline{y}_i, \underline{x}_i) > 1] = 1 - (1-q)^k - kq(1-q)^{k-1}$$

where $d_H(\underline{y}, \underline{x})$ is the Hamming distance between $\underline{y}$ and $\underline{x}$. To sum up, the basic version of the Winnow protocol should be complexified to compete with Cascade.
The best performances of Cascade, as the trade-off between the efficiency and the error correction capability, are achieved using $k_1 = 0.73/q$ and 2 iterations.

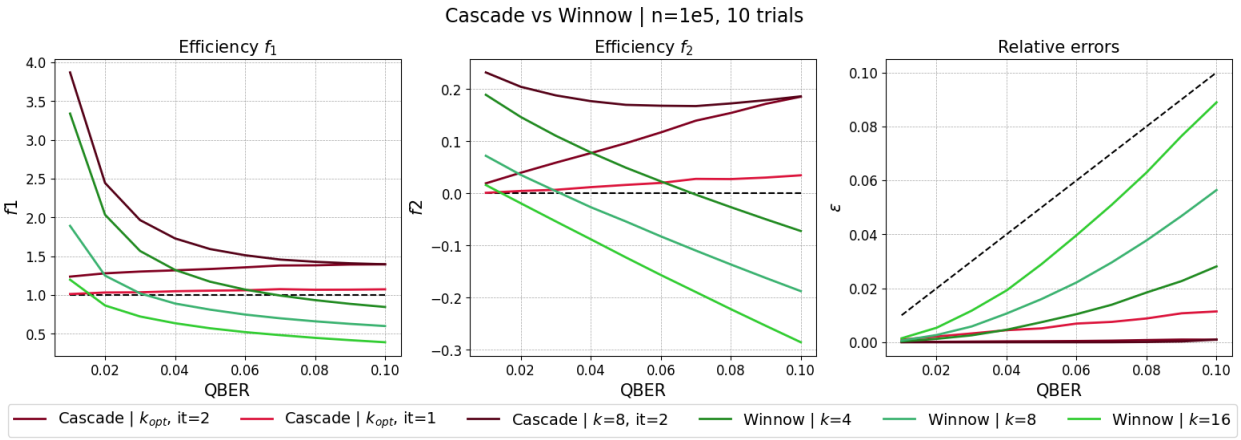

Figure 2. Performances of the Cascade and Winnow protocols.

For what concerns the LDPC codes, we used a $n = 1944$ code with four different code rates. We analyzed the efficiency of the protocol and the number of additional iterations needed to correctly reconcile the keys.

As we can see, using unmodulated codes, there is a trade-off between the additional iterations and the number of unnecessary disclosed bits ($f_1 >> 1$).

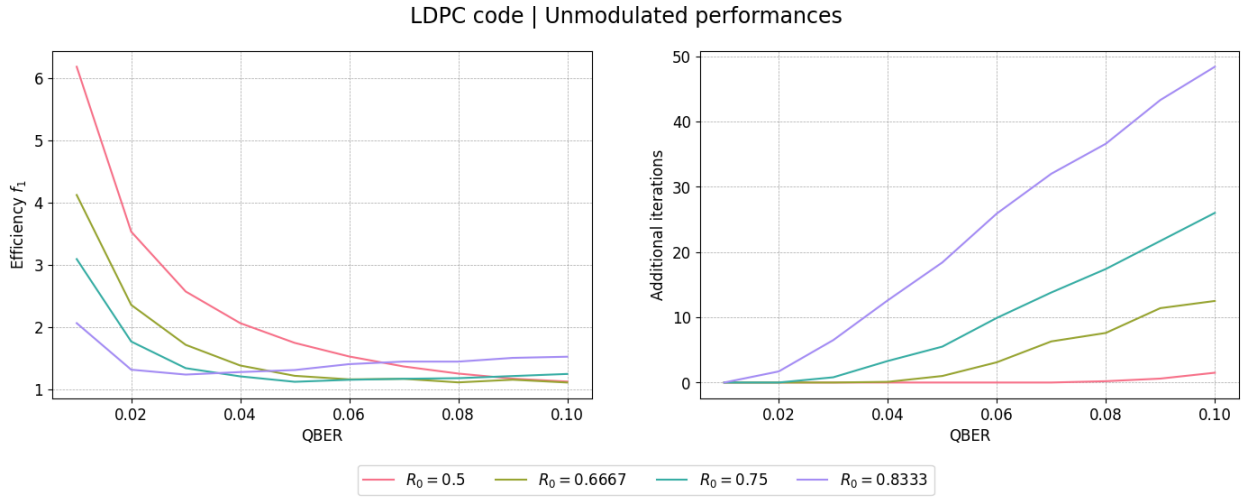

Figure 3. Performances of the unmodulated LDPC code.

Figure 4 describes the role of the rate modulation, in terms of shortening bits s, puncturing bits p, and the obtained rates. The first two parameters are common to all the initial rate R_0.

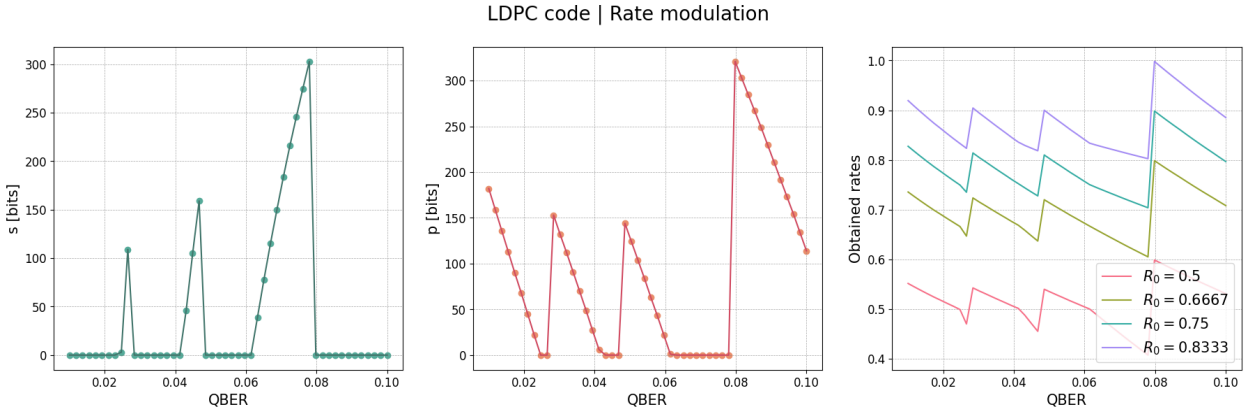

Figure 4. Optimal code rate modulation.

The code rate modulation is such as to make the performances equivalent both in terms of efficiency and additional iterations needed. Figure 5 reports the same features of figure 3 after the rate modulation.

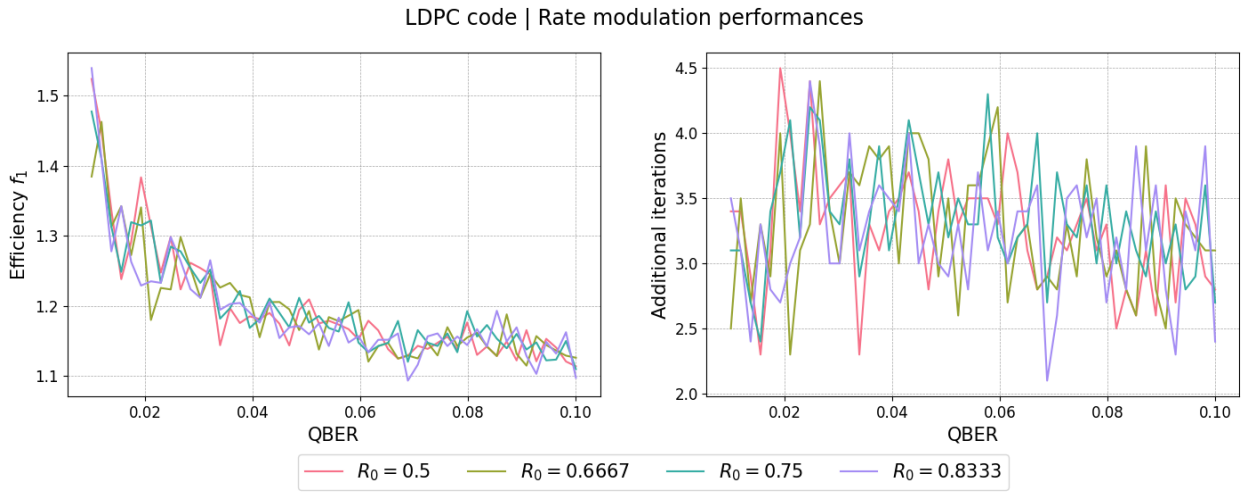

Figure 5. Performances of the LDPC code after rate modulation.

One can better understand the role of code modulation by manually changing the number of shortening and puncturing bits and evaluating the performances of the information reconciliation. The value of the QBER is crucial in the results, as we can see comparing figure 6 and figure 7. We repeated this analysis both in the case of a deterministic number of errors and for a binary symmetric channel (BSC) with - qualitatively - equivalent results.
Here are the results for a QBER of 2%:

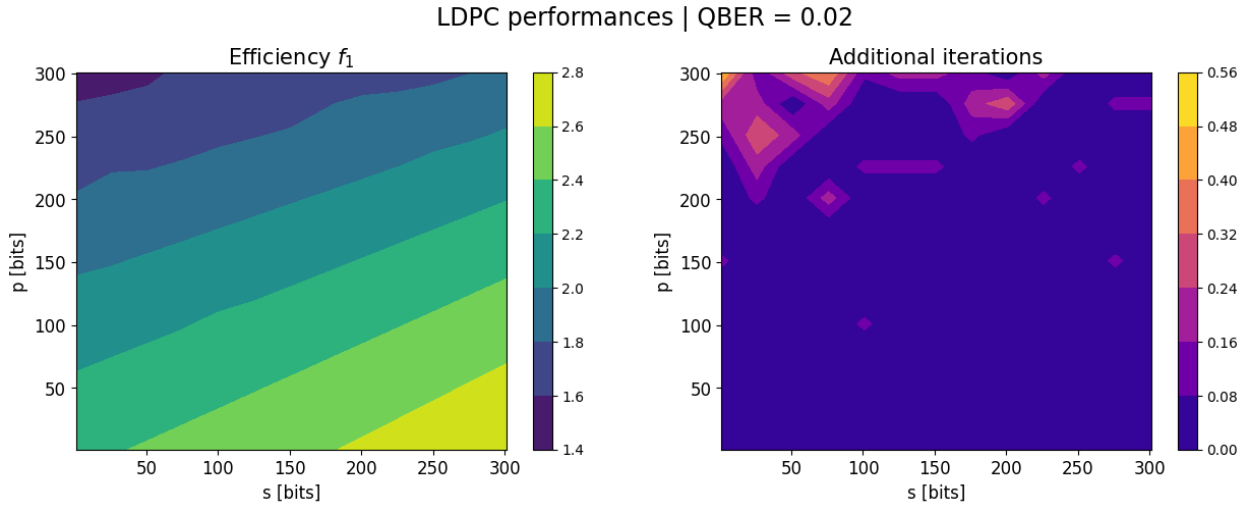

Figure 6. Role of s and p: low QBER, BSC, $R_0 = 0.6667$

Same analysis using a QBER of 7%:

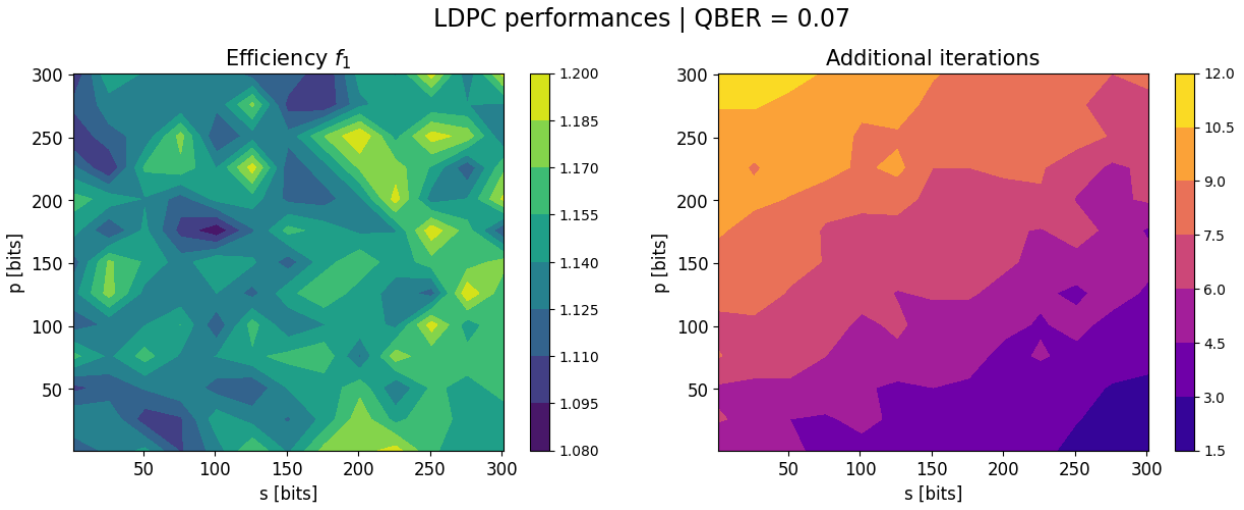

Figure 7. Role of s and p: high QBER, BSC, $R_0 = 0.6667$.

With low QBERs, the number of additional iterations is neglectable, and the best thing to do is to increase the rate of transmission by increasing the puncturing bits. On the other hand, to deal with high QBERs it's better to increase the number of shortening bits to decrease the number of required iterations.

Finally, we report the comparison between the efficiencies f_1 and f_2 of the Cascade protocol and the LDPC code, using the best cases of both methods.
The LDPC codes outreach Cascade for QBERs greater than 3%, proving to be an efficient method to reconcile the keys while limiting the information leakage.

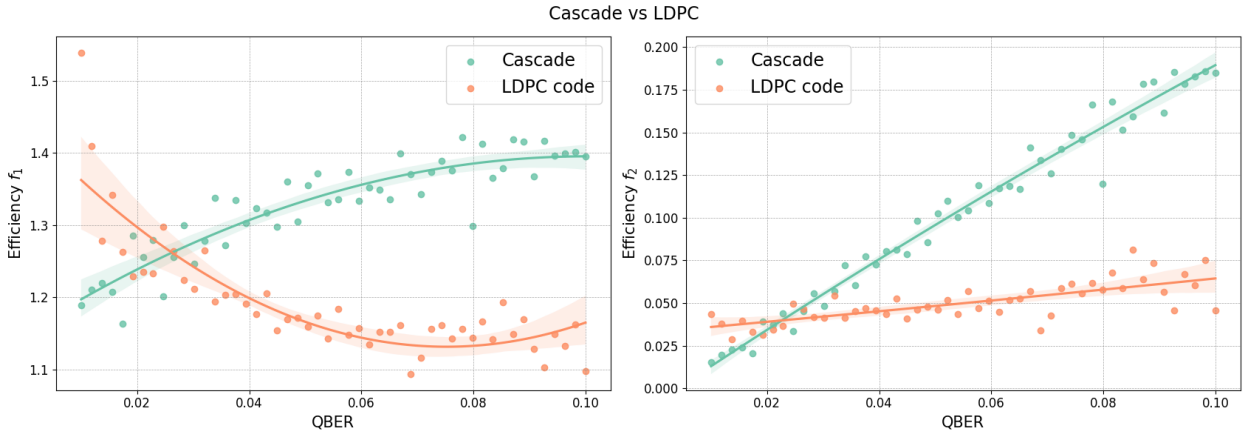

Figure 8. Performances of the Cascade (k_{opt}, it=2) and the modulated LDPC code.

4 Conclusions

We implemented some numerical simulations of three different algorithms used during the *information reconciliation* step of a quantum key distribution protocol. In particular, we compared Winnow, Cascade, and the LDPC coding in terms of two different efficiency metrics.
The basic version of Winnow should be complexified to compete with the other two protocols, both in terms of efficiency and error correction capability. On the other hand, Cascade and the LDPC codes achieved comparable results. However, the latter reaches even better performances due to the rate modulation and reduces the interactivity of Cascade, exchanging only one message by both authenticated parties.

References

[1] Miralem Mehic, Marcin Niemiec, Harun Siljak, and Miroslav Voznak. Error reconciliation in quantum key distribution protocols., 2020.

[2] Gilles Brassard and Louis Salvail. Secret-key reconciliation by public discussion. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 410–423. Springer, 1993.

[3] Tomohiro Sugimoto and Kouichi Yamazaki. A study on secret key reconciliation protocol. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 83(10):1987–1991, 2000.

[4] William T Buttler, Steven K Lamoreaux, Justin R Torgerson, GH Nickel, CH Donahue, and Charles G Peterson. Fast, efficient error reconciliation for quantum cryptography. *Physical Review A*, 67(5):052303, 2003.

[5] David Elkouss, Jesus Martinez-Mateo, and Vicente Martin. Information reconciliation for quantum key distribution. *arXiv preprint arXiv:1007.1616*, 2010.

[6] David Slepian and Jack Wolf. Noiseless coding of correlated information sources. *IEEE Transactions on information Theory*, 19(4):471–480, 1973.