Quantum Information Processing (2024) 23:195
https://doi.org/10.1007/s11128-024-04395-w

Efficient information reconciliation for high-dimensional quantum key distribution

Ronny Mueller[1] · Domenico Ribezzo[2,3] · Mujtaba Zahidy[1] · Leif Katsuo Oxenløwe[1] · Davide Bacco[4] · Søren Forchhammer[1]

Received: 6 November 2023 / Accepted: 17 April 2024 / Published online: 17 May 2024

Abstract
The information reconciliation phase in quantum key distribution has significant impact on the range and throughput of any QKD system. We explore this stage for high-dimensional QKD implementations and introduce two novel methods for reconciliation. The methods are based on nonbinary LDPC codes and the Cascade algorithm, respectively, and achieve efficiencies close the Slepian–Wolf bound on q-ary symmetric channels.

Keywords Quantum key distribution · Information reconciliation · High-dimensional · Cascade · LDPC

1 Introduction

Quantum key distribution (QKD) protocols allow for secure transmission of information between two entities, Alice and Bob, by distributing a symmetric secret key via a quantum channel [1, 2]. The process involves a quantum stage where quantum information is distributed and measured. This quantum stage is succeeded by post-processing. In this purely classical stage, the results of the measurements undergo a reconciliation process to rectify any discrepancies before a secret key is extracted during the privacy amplification phase. The emphasis of this work is on the phase of information reconciliation, which impacts the range and throughput of any QKD system.

✉ Ronny Mueller
ronmu@dtu.dk

1 Department of Electrical and Photonics Engineering, Technical University of Denmark, Lyngby, Denmark

2 National Institute of Optics of National Research Council, Florence, Italy

3 University of Naples Frederico II, Naples, Italy

4 Department of Physics and Astronomy, University of Florence, Florence, Italy

Despite the considerate development of QKD technology using binary signal forms, its high-dimensional counterpart (HD-QKD)[3] has seen significantly less research effort so far. However, HD-QKD offers several benefits, including higher information efficiency and increased noise resilience [4–7]. Although the reconciliation phase for binary-based QKD has been extensively researched, little work has been done to analyze and optimize this stage for HD-QKD, apart from introducing the layered scheme in 2013 [8]. This study addresses this research gap by introducing two novel methods for information reconciliation for high-dimensional QKD and analyzing their performance.

Unlike the majority of channel coding applications, the (HD)-QKD scenario places lesser demands on latency and throughput while emphasizing significantly the minimization of information leakage. Spurred by this unique setting, the strong decoding performance of nonbinary LDPC codes [9], and their inherent compatibility with high dimensions, we investigate the construction and utilization of nonbinary LDPC codes for post-processing in HD-QKD protocols as the first method.

The second method we investigate is the Cascade protocol [10]. It is one of the earliest proposed methods for reconciling keys. While many rounds of communication required by Cascade and concerns about resulting limitations on throughput have led to a focus on syndrome-based methods [11–13] in the past decade, recent research has shown that sophisticated software implementations can enable Cascade to achieve high throughput even with realistic latency on the classical channel [14, 15]. Motivated by these findings, we explore the usage of Cascade in the reconciliation stage of HD-QKD and propose a modification that enables high reconciliation efficiency for the respective quantum channel.

To the best of our knowledge, the only prior work investigating information reconciliation for high-dimensional QKD is the aforementioned layered scheme, introduced in 2013. The layered scheme is based on decoding bit layers separately using $\lceil \log_2(q) \rceil$ binary LDPC codes. It is similar in concept to the multilevel coding and multistage decoding methods used in slice reconciliation for continuous-variable (CV) QKD [16].

LDPC codes have been widely studied and optimized for the use in binary QKD systems [17–20] and can reach good efficiency with high throughput. The use of nonbinary LDPC codes has also been investigated for binary QKD [21]. Modifications of Cascade have been shown to achieve efficiency performance close to the theoretical limit [22] on binary QKD systems while simultaneously reaching high throughput [14]. Except for the layered scheme, neither LDPC codes, Cascade, nor any other error correction method has yet been optimized, modified or analyzed for the use in high-dimensional QKD.

This work has the following outline: In Sect. 2.1, the general scenario of Information Reconciliation is introduced. This is followed by an introduction of nonbinary LDPC codes and density evolution in Sect. 3.1. In Sect. 2.3, the usage of Cascade is reviewed and a novel algorithm for high-dimensional information reconciliation, high-dimensional Cascade, is introduced. The results, i.e., the performance of both nonbinary LDPC codes and high-dimensional Cascade, are shown in Sect. 3 followed by a discussion and comparison in Sect. 4. The work concludes in a short review of the achieved results in Sect. 5.

2 Background

In this section, we describe the general setting and channel model, and introduce relevant figures of merit. We then continue to describe the two proposed methods, nonbinary LDPC codes and high-dimensional Cascade, in more detail.

2.1 Information reconciliation

The goal of the information reconciliation stage in QKD is to correct any discrepancies between the keys of the two parties while minimizing the information leaked to potential eavesdroppers. Generally, Alice sends a random string $\mathbf{x} = (x_0, ..., x_{n-1})$, $x_i = 0, ..., q-1$ of n qudits of dimension q to Bob, who measures them and obtains his version of the string $\mathbf{y} = (y_0, ..., y_{n-1})$, $y_i = 0, ..., q-1$. A practical example of a qudit can be seen in Fig. 1. We assume that the quantum channel can be accurately represented by a substitute channel where $\mathbf{x}$ and $\mathbf{y}$ are correlated as a q-ary symmetric channel since errors are typically uncorrelated and symmetric. The transition probabilities of such a channel are as follows:

$$\mathrm{P}(y_i|x_i) = \begin{cases} 1-p & y_i = x_i, \\ \frac{p}{q-1} & \text{else.} \end{cases} \tag{1}$$

Here, the parameter p represents the channel transition probability. We refer to the symbol error rate between $\mathbf{x}$ and $\mathbf{y}$ as the quantum bit error rate (QBER) in a slight abuse of notation but consistent with experimental works on HD-QKD. In our simulations, we assume the QBER to be an inherent channel property, making it equivalent to the channel parameter p. In addition to the qudits, Alice also sends classical messages, e.g., syndromes or parity bits, which are assumed to be error-free. From a coding perspective, this is equal to asymmetric Slepian–Wolf coding with side information at the receiver, where the syndrome $\mathbf{s}$ represents the compressed version of $\mathbf{x}$, and $\mathbf{y}$ is the side information. A more detailed explanation of this equivalence can be found in [23]. For an interpretation of Cascade in the context of linear block codes see [22]. Any information leaked to a potential eavesdropper at any point during the quantum key distribution must be subtracted from the final secret key during privacy amplification [24]. The information leaked during the information reconciliation stage will be denoted by leak_{IR}. In the case of LDPC codes, assuming no rate adaptation, it can be upper-bounded by the syndrome length in bits, $\text{leak}_{\text{IR}} \le m$, with m being the

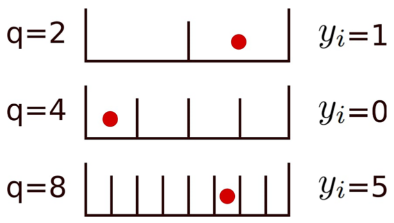


Fig. 1 Example of a qudit using a time-bin implementation [31]. The dimension of the qudit is set by the number of bins grouped together, while the value is determined by the measured arrival time

length of a binary representation of the syndrome string. In the case of Cascade, it can be upper-bounded by the number of parity bits sent from Alice to Bob [25], although attention has to be paid to special cases in relation to the parameter estimation phase of QKD post-processing [26]. Using the Slepian–Wolf bound [27], the minimum amount of leaked information required to successfully reconcile with an arbitrarily low failure probability in the asymptotic limit of infinite length is given by the conditional entropy:

$$\text{leak}_{\text{IR}} \geq n\text{H}(X|Y). \tag{2}$$

The conditional entropy (base q) of the q-ary symmetric channel, assuming independent and identically distributed input X, can be expressed as

$$\text{H}(X|Y) = -((1-p)\log_q(1-p) + p \cdot \log_q(\frac{p}{q-1})). \tag{3}$$

A code's performance in terms of relative information leakage can be measured by its efficiency f, given by

$$f = \frac{\text{leak}_{\text{IR}}}{n\text{H}(X|Y)}. \tag{4}$$

It is important to note that an efficiency of $f > 1$ corresponds to leaking more bits than required by the theoretical minimum of $f = 1$, which represents the best possible performance according to the Slepian–Wolf bound. In practice, systems have $f > 1$ due to the difficulty of designing optimal codes, finite-size effects, and the inherent trade-off between efficiency and throughput. For more details on achievable information leakage, including respect to finite-size effects, see, for example, [28]. In the following sections, we restrict ourselves to q being a power of 2. Both approaches can function without this restriction, but it allows for more efficient implementation of the reconciliation and is commonly seen in physical implementations of the quantum stage due to symmetries.

The information reconciliation phase is succeeded by an Error Verification stage wherein an estimate of the expected probability of correctness is obtained. Here, correctness refers to the agreement of both Alice's and Bob's versions of the key after information reconciliation. In practical terms, this is frequently accomplished through the exchange and comparison of hashes [29, 30]. In case of a disagreement, the error correction can be repeated with the cost of doubling the leaked information. If this is not feasible, e.g., the additional leakage prohibits any secret key extraction, the keys of this round are discarded.

2.2 Nonbinary LDPC codes

2.2.1 Codes & decoding

We provide here a short overview over nonbinary LDPC codes and their decoding based on the concepts and formalism of binary LDPC codes. For a comprehensive review of those, we refer to [32].

Nonbinary LDPC codes can be described by their parity check matrix $\mathbf{H}$, with m rows and n columns, containing elements in a Galois Field (GF) of order q. To enhance clarity in this section, all variables representing a Galois field element will be marked with a hat, for instance, $\hat{a}$. Moreover, let $\oplus$, $\ominus$, $\otimes$, and $\oslash$ denote the standard operations on Galois field elements: Addition, subtraction, multiplication, and division [33]. An LDPC code can be depicted as a bipartite graph, known as the Tanner graph. In this graph, the parity check equations form one side, called check nodes, while the codeword symbols represent the other side, known as variable nodes. The Tanner graph of a nonbinary LDPC code also has weighted edges between check and variable nodes, where each weight corresponds to the respective entry of $\mathbf{H}$. The syndrome $\mathbf{s}$ of the q-ary string $\mathbf{x}$ is computed as $\mathbf{s} = \mathbf{Hx}$.

For decoding, we employ a log-domain FFT-SPA [34, 35]. In-depth explanations of this algorithm can be found in [36, 37], but we provide a summary here for the sake of completeness. Let Z represent a random variable taking values in GF(q), such that $\mathrm{P}(Z_i = k)$ indicates the probability that qudit i has the value $k = 0, ..., q-1$. The probability vector $\mathbf{p} = (p_0, ...p_{q-1})$, $p_j = \mathrm{P}(Z = j)$ can be converted into the log-domain using the generalized equivalent of the log-likelihood ratio (LLR) in the binary case, $\mathbf{m} = (m_0, ..., m_{q-1})$, $m_j = \log \frac{\mathrm{P}(Z=0)}{\mathrm{P}(Z=j)} = \log(\frac{p_0}{p_j})$. Unless specified otherwise, the logarithm is taken to base e. Given the LLR representation, probabilities can be retrieved through $p_j = \exp(-m_j) / \sum_{k=0}^{q-1} \exp(-m_k)$. We use $p(\cdot)$ and $m(\cdot)$ to denote these transforms. To further streamline notation, we define the multiplication and division of an element $\hat{a}$ in GF(q) and an LLR message as a permutation of the indices of the vector:

$$\hat{a} \cdot \mathbf{m} := (m_{\hat{0} \otimes \hat{a}}, ..., m_{q \hat{-} 1 \otimes \hat{a}}) \tag{5}$$

$$\mathbf{m}/\hat{a} := (m_{\hat{0} \oslash \hat{a}}, ..., m_{q \hat{-} 1 \oslash \hat{a}}), \tag{6}$$

where the multiplication and division of the indices occur in the Galois Field. These permutations are necessary as we need to weigh messages according to their edge weight during decoding. We further define two transformations involved in the decoding,

$$\mathcal{F}(\mathbf{m}, \hat{H}ij) = \mathcal{F}(p(\hat{H}ij \cdot \mathbf{m})) \tag{7}$$

$$\mathcal{F}(\mathbf{m}, \hat{H}ij)^{-1} = m(\mathcal{F}^{-1}(\mathbf{m}))/\hat{H}ij, \tag{8}$$

where $\mathcal{F}$ represents the discrete Fourier transform. Note that for q being a power of 2, the fast Walsh Hadamard transform can be utilized. The decoding process then consists of two iterative message-passing phases, from check nodes to variable nodes and vice versa. The message update rule at iteration l for the check node to variable node message corresponding to the parity check matrix entry at (i, j) can be expressed as

$$\mathbf{m}_{ij,\mathrm{CV}}^{(l)} = \mathcal{A}(\hat{s}_i')\mathcal{F}^{-1}(\prod_{j' \in \mathcal{M}(i)/j} \mathcal{F}(\mathbf{m}_{ij'}^{(l-1)}, \hat{H}_{ij'}), \hat{H}_{ij}), \tag{9}$$

where $\mathcal{M}(i)$ denotes the set of all check nodes in row i of $\mathbf{H}$. The matrix $\mathcal{A}$, defined as $\mathcal{A}_{kj}(\hat{a}) = \delta(\hat{a} \oplus k \ominus j) - \delta(a \ominus j)$, accounts for the possible occurrence of nonzero syndromes [37]. The weighted syndrome value is calculated as $\hat{s}'_i = \hat{s}_i \oslash \hat{H}_{ij}$. The a posteriori message of column j can be written as

$$\tilde{\mathbf{m}}_j^{(l)} = \mathbf{m}^{(0)}(j) + \sum_{i' \in \mathcal{N}(j)} \mathbf{m}_{i'j,\mathrm{CV}}^{(l)}, \tag{10}$$

where $\mathcal{N}(j)$ is the set of all check nodes in column j of $\mathbf{H}$. The best guess $\tilde{\mathbf{x}}$ at each iteration l can be calculated as the minimum value of the a posteriori, $\tilde{x}_j^{(l)} = \operatorname{argmin}(\tilde{\mathbf{m}}_j^l)$. The second message passings, from variable to check nodes, are given by

$$\mathbf{m}_{ij,\mathrm{VC}}^{(l)} = \tilde{\mathbf{m}}_j^{(l)} - \mathbf{m}_{ij,\mathrm{CV}}^{(l)}. \tag{11}$$

The message passing continues until either $\mathbf{H}\tilde{\mathbf{x}} = \mathbf{s}$ or the maximum number of iterations is reached.

To allow for efficient reconciliation for different QBER values, a rate-adaptive scheme is required. We use the blind reconciliation protocol [11] as it has a better performance with respect to efficiency than direct rate adaption [38]. A fixed fraction δ of symbols is chosen to be punctured or shortened. Puncturing refers to replacing a key bit with a random bit that is unknown to Bob. For shortening, the value of the bit is additionally sent to Bob over the public channel. Puncturing, therefore, increases the code rate, while shortening lowers it. The rate of a code with p punctured and s shortened bits is then given by

$$R = \frac{n - m - s}{n - p - s}. \tag{12}$$

To see how rate adaption influences the bounding of $\mathrm{leak}_{\mathrm{IR}}$, see [39]. The blind scheme introduces interactivity into the LDPC reconciliation. Given a specific code, we start out with all bits being punctured and send the respective syndrome to Bob. Bob attempts to decode using the syndrome. If decoding fails, Alice transforms $\lceil n(0.028 - 0.02R) \rceil$ [40] punctured bits into shortened bits, and resends the syndrome. This value is a heuristic expression and presents a trade-off between the number of communication rounds and the efficiency. Bob tries to decode again and requests more bits to be shortened in case of failure. If there are no punctured bits left to be turned into shortened bits, Alice reveals plain key bits instead. This continues until either decoding succeeds or the whole key is revealed by Alice successively sending all key bits through the public channel.

2.2.2 Density evolution

In the case of a uniform edge weight distribution, the asymptotic decoding performance of LDPC codes for infinite code length is entirely determined by two polynomials [41, 42]:

$$\lambda(x) = \sum_{i=0}^{d_{\mathrm{v,\,max}}} \lambda_i x^{i-1} \quad \rho(x) = \sum_{i=0}^{d_{\mathrm{c,\,max}}} \rho_i x^{i-1}. \tag{13}$$

In these expressions, λ_i (ρ_i) represents the proportion of edges connected to variable (check) nodes with degree i, while $d_{\mathrm{v,max}}$ ($d_{\mathrm{c,max}}$) indicates the highest degree of the variable (check) nodes. Given these polynomials, we can then define the code ensemble $\mathcal{E}(\lambda, \rho)$, which represents all codes of infinite length with degree distributions specified by λ and ρ. The threshold $p_t(\lambda, \rho)$ of the code ensemble $\mathcal{E}(\lambda, \rho)$ is defined as the worst channel parameter (QBER) at which decoding remains possible with an arbitrarily small failure probability. This threshold can be estimated using Monte Carlo density evolution (MC-DE), which is thoroughly described in [43]. This technique repeatedly samples node degrees according to λ and ρ, and draws random connections between nodes for each iteration. With a sufficiently large sample size, this simulates the performance of a cycle-free code. Note that MC-DE is particularly well suited for nonbinary LDPC codes, as the distinct edge weights aid in decorrelating messages [43]. During the simulation, we track the average entropy of all messages [43]. When it falls below a certain value, decoding is considered successful. If this does not occur after a maximum number of iterations, the evaluated channel parameter is above the threshold of $\mathcal{E}(\lambda, \rho)$. Utilizing a concentrated check node distribution (which is favorable according to [44]) and a fixed code rate, we can further simplify to $\mathcal{E}(\lambda)$. The threshold can then be employed as an objective function to optimize the code design, which is commonly achieved using the differential evolution algorithm [45].

2.3 Cascade

2.3.1 Binary cascade

Cascade [10] is one of the earliest schemes proposed for information reconciliation and has seen widespread use due to its simplicity and high efficiency. Successive works on the original Cascade protocol have been trying to increase its performance by either substituting the parity exchange with error correction methods [46], or by optimizing parameters like the top-level block sizes [47]. The binary Cascade protocol acting on a single frame can be summarized in the following steps:

- Iteration 1:
  1. The binary frame is divided into non-overlapping blocks of size k_1, where the value of k_1 usually depends on the estimated QBER of the frame.
  2. Alice and Bob calculate the parity of each top-level block and share them over a classical channel.
  3. For those blocks where a mismatch between the parity of Alice and the parity of Bob is detected, a binary search is used to detect a single error. In general, iff a parity mismatches between Alice and Bob, a binary search can be performed on that block. The binary search consists of three steps:
     1. Split the respective block in half.
     2. Calculate and exchange the parities of the two sub-blocks. One of the sub-blocks has a mismatching parity.

3. If the mismatching sub-block contains only 1 bit, the error is found. Otherwise, repeat Step (a) with the mismatching sub-block.

- Iteration i:
  1. Apply a permutation on the frame and divide it into new top-level blocks of size k_i. Repeat Steps 2 and 3 as described for the first iteration on the new blocks.
  2. Cascade step: Any erroneous bit detected in iteration i also takes part in blocks created in previous iterations. After correcting those bits, their parity mismatches again and allows for the detection of another error using binary search. This error again takes part in blocks of all other iterations and can be used to detect more errors, creating the ”cascading" effect of the Cascade step.

The protocol stops after a fixed number of iterations. To the best of our knowledge, state-of-the-art in terms of efficiency with values up to $f = 1.025$ is reached by a modification [22] of the original Cascade. The additions of this modification constitute mainly of separating bits into groups of similar confidence and choosing optimal block sizes on those groups in Iteration 2. We will denote this version of Cascade as “binary Cascade” and it will be used for all comparisons. It has also been used as the base for a recent implementation reaching the highest throughout so far of 570 Mbps [14].

2.3.2 High-dimensional Cascade

We propose the following modification to use Cascade for high-dimensional data, which we will denote by high-dimensional Cascade (HD-Cascade). We only highlight the differences compared to binary Cascade as described in [22]. All the modifications we propose are additions to the binary algorithm and reduce back to binary Cascade for $q = 2$ as they rely on correlations that only exist for $q > 2$.

- *Modification 1* Initially, we map all symbols to an appropriate binary representation. Prior to the first iteration, we shuffle all bits while maintaining a record of which bits originate from the same symbol. We denote these bits as ”partner bits". This mapping effectively reduces the expected QBER used for block size calculations in all iterations to $\mathrm{QBER}_{\mathrm{BIN}} = \frac{q}{2(q-1)}\mathrm{QBER}_{\mathrm{SYM}}$.

- *Modification 2* Upon detecting an error at any point during the protocol, immediately do the following:
  1. Request all partner bits of the erroneous bit.
  2. If any of the partner bits are erroneous, the blocks they have been participating in now have mismatching parities again. Run a binary search on all mismatching blocks. Repeat Step 1 until no new errors are found.

  The conditional probability for a partner bit to be erroneous given the values of all previously transmitted bits for these bits is close to $1/2$. To be precise, it is equal to $1/2$ for bits that have not yet participated in any parity checks and then varies with the length of the smallest block they participated in [22]. Note that this procedure allows for a cascading process in the first iteration already, and therefore requires

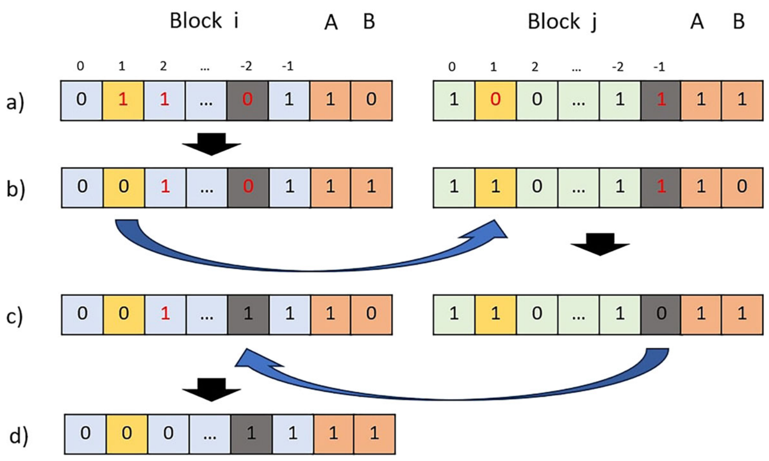


Fig. 2 Example of the cascading step inside the first iteration of HD-Cascade. Blocks i and j are random top-level blocks in the first iteration. The bits at positions $(i, 1)$ & $(j, 1)$ and $(i, -2)$ & $(j, -1)$ originate from the same symbol, respectively. A and B denote the current parity of the block for Alice and Bob, respectively. **a** Block j has a matching parity, so no binary search is possible. Block i has mismatching parity, a binary search reveals position 1 to be erroneous. **b** After correcting the error, Bob's parity flips to match that of Alice. By requesting and correcting the associated partner bit in block j, a mismatch in the parities is introduced. We can therefore run a binary search on block j and detect the error at position $(j, -1)$. **c** By requesting and correcting the respective partner bit in block i, the parities mismatch again and allow for another round of binary search to reveal the error at position $(i, 2)$. **d** All blocks have matching parities

all blocks to be processed sequentially followed by a Cascade step for each single error for maximum impact of the Cascade step.

- *Modification 3* The fraction of errors corrected in the first iteration is significantly higher (often $> 95\%$ in our simulations for high dimensions) compared to the binary version. This is due to the possibility of running a cascading process in the first iteration already, see Fig. 2 for an example. Consequently, we need to increase the block sizes for the following iterations as the dimensionality increases, see Table 2. The importance of partner bits increases with increasing dimension.

2.3.3 Parallel high-dimensional Cascade

While the proposition in Sect. 2.3.2 achieves great efficiency, it also requires all blocks to be processed serially. This results in a limited throughput, as a large amount of messages is required, i.e., a single message for every requested parity. We therefore propose the following adaptions for a more practical implementation of high-dimensional Cascade:

1. *Modification 4* In the first iteration, we split the binary representation into $v = \log_2(q)$ groups of size n; n being the number of qudits per frame. Each group only contains bits of the same bit-plane, i.e., the first group contains all the first bits of all symbols, the second group contains all the second bits of all symbols and so on. We then apply permutations that are restricted to each group only. After locating

the error positions of one group using binary search, the corresponding partner bits will be located in all other groups. All blocks of one group can therefore be processed in parallel. When calculating the top-level block sizes of the following blocks, we adjust the expected QBER using the number of partner bits. It can be calculated for group i, $i = 1, ..., v$ as:

$$\mathrm{QBER}_i = \mathrm{QBER}_{\mathrm{BIN}} - \frac{1}{2n} \sum_{j<i} \frac{\mathrm{PB}_j}{v}, \qquad (14)$$

where PB_j denotes the number of partner bit requests originating from group j. To reduce the number of messages sent, we do not cascade on the partner bits until all groups have finished this first stage. The block creation for all other iterations follows the procedure described in [22].

2. *Modification 5* This modification replaces Modification 2 and the Cascade step of the binary protocol. After completing the binary search for all top-level blocks of an iteration, we collect all found errors into a list. We then do the following for the list of known errors:
   (a) All known error positions are fed into the Cascade step as a group and processed in parallel. For each error, locate the smallest block for which the error participates in a not-flagged iteration.
   (b) Run a binary search on those smallest blocks that have a mismatch and locate the new error positions. Add the new errors to the list of all errors. Request the partner bits of these new errors if not already known and add them to the list of all known errors if they are erroneous. Flag the origin iterations of the blocks as already used for the respective input bits. The origin iteration of a block is that iteration in which the block has been created, either as a top-level block or as part of the binary search.
   (c) Remove all errors from the list that have all iterations flagged. Correct all known errors and repeat Step (a). If there are no new errors to be found, the Cascade step terminates.

   The Cascade step can be stopped early in a trade-off between efficiency, throughput, and frame error rate.

3 Results

3.1 Nonbinary LDPC codes

While the code design and decoding techniques described above are feasible for any dimension q, we focus on $q = 4$ and 8 as those are common in current implementations [48]. Nine codes were designed with code rates between 0.50 and 0.90 for $q = 4$ ($q = 8$), corresponding to a supported QBER range between 0 and 18% (24.7%). We used 100000 nodes with a maximum of 150 iterations for the MC-DE, the QBER was swept in 20 steps in a short range below the best possible threshold. In the differential evolution, population sizes between 15 and 50, a differential weight of 0.85, and a crossover probability of 0.7 were used. A sparsity of at most 10 nonzero coefficients in the polynomial was enforced, with the maximum node degree chosen as $d_{\mathrm{v,max}} = 40$. The sparsity allowed for reasonable optimization complexity, the maximum node degree was chosen to avoid numerical instability which we observed for higher values.

The results of the optimization can be found in Table 1 in form of the node degree distributions (13) and their performance according to density evolution by reporting their simulated threshold (density evolution threshold, DET). The efficiency was evaluated for the highest supported QBER, noted as the ensemble efficiency (EEff). The all-zero codeword assumption was used for the optimization and evaluation, which holds for the given scenario of a symmetric channel [36]. For all rates, the designed thresholds are close to the theoretical bound (2). LDPC codes with a length of $n = 30000$ symbols were constructed using Progressive Edge Growth [49], and a log-FFT-SPA decoder was used to reconcile the messages. The simulated performance of the finite-size codes can be seen in Fig. 6 for a span of different QBER values, each data point being the mean of 100 samples. We used the blind reconciliation scheme for rate adaption [11]. The mean number of decoding tries required for Bob to successfully reconcile is also shown. The valley pattern visible in the efficiency of the LDPC codes is due to the transition between codes of different rate, and a slight degradation in performance for high ratios of puncturing or shortening. The decoder used a maximum of 100 decoding iterations. As expected for finite-size codes, they do not reach the asymptotic ensemble threshold but show sub-optimal performance [37].

3.2 High-dimensional Cascade

The performance of HD-Cascade, see Sect. 2.3.2, was evaluated on the q-ary symmetric channel for dimensions $q = 4, 8, 32$, and for a QBER ranging from 1% to 20%. The results are shown in Fig. 3. For comparison, a direct application of the best-performing Cascade modification on a binary mapping is also included. Binary Cascade's performance can be understood as generating an information leakage corresponding to an input key with an error rate of $\mathrm{QBER}_{\mathrm{BIN}}$. The proposed high-dimensional Cascade uses the same base Cascade with the additional adaptations discussed in Sect. 2.3.2. For $q = 2$, HD-Cascade reduces to binary Cascade, resulting in equal performance. Both methods use the same block size of $n = 2^{16}$ bits for all cases. The used top-level block sizes k_i for each iteration i can be seen in Table 2, where $[\cdot]$ denotes rounding to the nearest integer. The block sizes have been chosen heuristically through numerical optimization. Additionally, the layered scheme is included as a reference [8]. All data points have a frame error rate below 1% and show an average of 1000 samples. The wave pattern observable for the efficiency of Cascade in Fig. 3 and Fig. 6 is due to the integer rounding operation when calculating the block sizes. Discretized block sizes being a power of two have been shown to be optimal for the binary search in this setting [22].

The increase in both the range and secret key rate resulting from using HD-Cascade instead of directly applying binary Cascade is depicted in Fig. 4. The used protocols

Table 1 Degree distributions for four- and eight-dimensional nonbinary LDPC codes

	4-dimensional		
Rate	DET	EEff	Ensemble (edge view)
0.90	0.022	1.067	$\lambda(x) = 0.069x + 0.207x^2 + 0.051x^5 + 0.235x^6 + 0.070x^{14} + 0.225x^{17}$ $+0.139x^{28}$
0.85	0.037	1.045	$\lambda(x) = 0.094x + 0.210x^2 + 0.217x^6 + 0.088x^9 + 0.040x^{10} + 0.118x^{22}$ $+0.159x^{24} + 0.071x^{28}$
0.80	0.053	1.044	$\lambda(x) = 0.102x + 0.204x^2 + 0.116x^5 + 0.088x^7 + 0.174x^{14} + 0.107x^{26}$ $+0.205x^{27}$
0.75	0.069	1.053	$\lambda(x) = 0.107x + 0.245x^3 + 0.192x^6 + 0.034x^9 + 0.207x^{18} + 0.161x^{25}$ $+0.049x^{27}$
0.70	0.08	1.054	$\lambda(x) = 0.113x + 0.245x^2 + 0.143x^4 + 0.081x^{10} + 0.066^{14} + 0.147^{16}$ $+0.034x^{23} + 0.168x^{24}$
0.65	0.11	1.045	$\lambda(x) = 0.133x + 0.213x^2 + 0.207x^5 + 0.014x^8 + 0.022x^{17} + 0.171x^{19}$ $+0.237x^{27}$
0.60	0.13	1.047	$\lambda(x) = 0.172x + 0.252x^2 + 0.216x^6 + 0.022x^{11} + +0.075x^{12} + 0.077x^{18}$ $+0.183x^{20}$
0.55	0.15	1.041	$\lambda(x) = 0.177x + 0.279x^2 + 0.147x^7 + 0.035x^8 + +0.088x^{10} + 0.129x^{14}$ $+0.143x^{25}$
0.50	0.18	1.037	$\lambda(x) = 0.184x + 0.245x^2 + 0.087x^6 + 0.156x^7 + +0.067x^{17} + 0.020^{21}$ $+0.196^{25} + 0.041^{28}$

Table 1 continued

Rate	4-dimensional		
	DET	EEff	Ensemble (edge view)
	8-dimensional		
0.90	0.031	1.060	$\lambda(x) = 0.112x + 0.103x^2 + 0.194x^3 + 0.146x^9 + 0.163x^{10} + 0.003x^{17}$ $+0.173x^{19} + 0.049x^{26} + 0.006x^{28} + 0.052x^{29}$
0.85	0.052	1.080	$\lambda(x) = 0.125x + 0.165x^2 + 0.163x^5 + 0.11x^7 + 0.073x^{12} + 0.089x^{18}$ $+0.122x^{27} + 0.154x^{32}$
0.80	0.072	1.038	$\lambda(x) = 0.146x + 0.177x^2 + 0.130x^4 + 0.084x^7 + 0.149x^{10} + 0.035x^{19}$ $+0.029x^{22} + 0.087x^{25} + 0.163x^{26}$
0.75	0.096	1.030	$\lambda(x) = 0.165x + 0.192x^2 + 0.092x^5 + 0.176x^7 + 0.019x^{10} + 0.086x^{17}$ $+0.129x^{19} + 0.103x^{30} + 0.038x^{31}$
0.70	0.121	1.032	$\lambda(x) = 0.160x + 0.208x^2 + 0.140x^5 + 0.096x^8 + 0.028x^{10} + 0.013x^{11}$ $+0.113x^{18} + 0.032x^{21} + 0.211x^{27}$
0.65	0.147	1.032	$\lambda(x) = 0.173x + 0.228x^2 + 0.092x^4 + 0.169x^8 + 0.112x^{14} + 0.019x^{23}$ $+0.012x^{24} + 0.195x^{28}$
0.60	0.177	1.026	$\lambda(x) = 0.192x + 0.196x^2 + 0.222x^5 + 0.104x^{13} + 0.114x^{23} + 0.055x^{25}$ $+0.117x^{27}$
0.55	0.207	1.024	$\lambda(x) = 0.183x + 0.269x^2 + 0.124x^6 + 0.036x^8 + 0.097x^{10} + 0.004^{21}$ $+0.116x^{25} + 0.171x^{26}$
0.50	0.239	1.024	$\lambda(x) = 0.215x + 0.256x^2 + 0.030x^4 + 0.154x^7 + 0.065x^{11} + 0.050^{13}$ $+0.072x^{21} + 0.128x^{27}$

The threshold calculated by density evolution (DET), the corresponding ensemble efficiency (EEff) and the node degree distribution in edge view

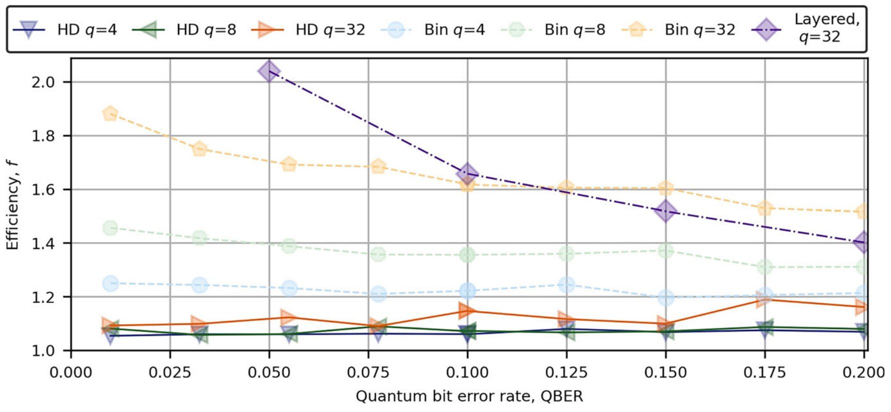


Fig. 3 Efficiency of different approaches evaluated on a q-ary symmetric channel. Layered refers to the layered scheme, Bin to direct application of binary Cascade (serial), and HD to the high-dimensional Cascade proposed in this work. Data points represent the mean of 1000 samples and have a FER of less than 1%

Table 2 Block sizes used for HD-Cascade

k_1							$\min(2^{[\log_2(1/\mathrm{QBER_{BIN}})]}, n/2)$
k_2							$\min(2^{[\log_2(2q/\mathrm{QBER_{BIN}})]}, n/2)$
k_3	k_4	k_5	k_6	$n/16$	$n/8$	$n/4$	$n/2$

are 1-decoy state QKD protocols [50, 51], with the secret key length l_q per block given as

$$l_q \le \log_2(q)D_0^Z + D_1^Z(\log_2(q) - \mathrm{H_{HD}}(\Phi_Z, q)) - \mathrm{leak_{IR}} - 6\log_2(19/\epsilon_{\mathrm{sec}}) - \log_2(2/\epsilon_{\mathrm{cor}}), \tag{15}$$

where D_0^Z is a lower bound on vacuum events and D_1^Z is a lower bound on single photon events. We refer to the supplementary information of [50] for derivation of these bounds. $\mathrm{H_{HD}}(\Phi_Z, q)$ is the high-dimensional Shannon entropy,

$$\mathrm{H_{HD}}(\Phi_Z, q) = -\Phi_Z \log_2(\Phi_Z/(q-1)) - (1-\Phi_Z)\log_2(1-\Phi_Z), \tag{16}$$

where Φ_Z is an upper bound on the phase error rate, ϵ_{sec} is the security parameter, and ϵ_{cor} is the correctness parameter. Experimental parameters for the simulation are derived from [52] for $q = 2$ and 4, where a combination of polarization and path is used to encode the qudits. For $q = 8$ and 32, we used a generalization of the setup. Additional losses might transpire due to increased experimental complexity which are not considered in the simulation. Some of the parameters are listed in Table 3.

The improvement in the relative secret key rate r obtained using HD-Cascade is shown in Fig. 7. We also analyzed the performance of HD-Cascade on experimental data provided by the experiment in [52], which confirms the simulated performance.

Table 3 Parameters used for the SKR simulations

ϵ_{sec}	Block-size privacy amplification	Repetition rate	Dead time	Insertion loss Z
10^{-12}	10^8	487 MHz	25 ns	1 dB
ϵ_{cor}	Misalignment error	Detector efficiency	Dark counts	Insertion loss X
10^{-12}	1%	86%	330	3 dB

Repetition rate refers to the repetition rate of the source, dead time to the dead time of the detectors used, and the insertion loss refers to the loss of the receiver side for the two bases used, X and Z. The dark counts and detector efficiency are given for a single detector. For more details see [50–52]

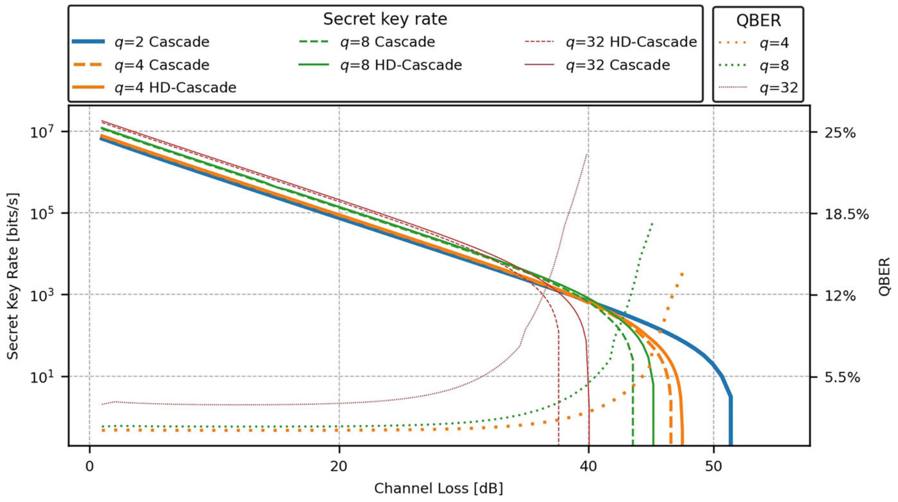


Fig. 4 Secret key rate vs. channel loss for different dimensions. The decreasing dotted/dashed lines show results for direct application of binary Cascade whereas the solid lines show the performance for HD-Cascade. The increasing dotted/dashed lines show the respective QBER

We further analyze the performance of the parallel high-dimensional Cascade implementation for $q = 4, 16$. The results can be seen in Fig. 5. While a slight penalty in efficiency can be observed, the number of messages sent for a single frame is greatly reduced compared to the serial approach. In the serial approach, the number of messages is roughly given by $n\text{H}(X|Y)$, i.e., around 8000 messages for $q = 16$, QBER$= 5\%$, and $f = 1.05$, compared to a mean of 239 and 189 messages for $q = 4$ and $q = 16$ for the parallel implementation. Notably, the number of messages seems to decrease for higher dimensions. All data points represent the mean of 2000 samples, the frame error rate is below 0.1% for all QBER values.

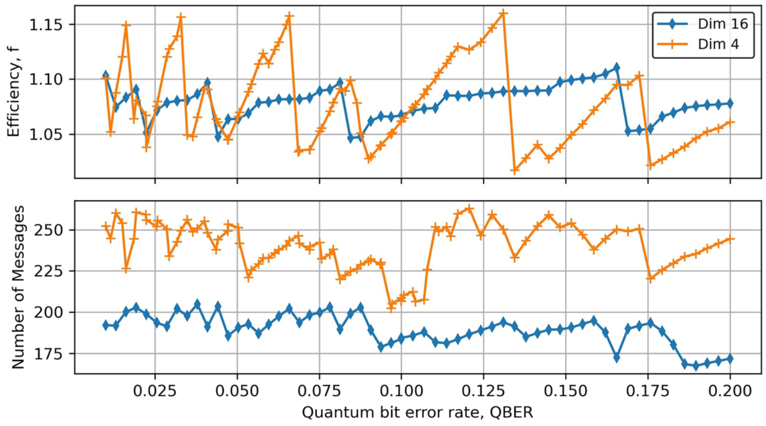
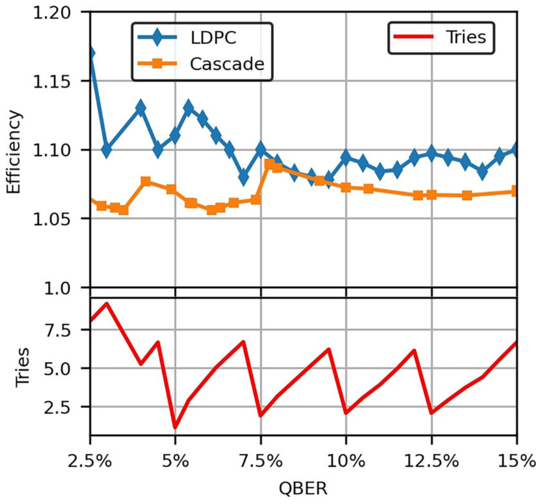


Fig. 5 Top: The efficiency of the parallel high-dimensional Cascade implementation for dimensions 4 and 16. Bottom: The number of messages sent from Alice to Bob for a single frame

Fig. 6 Top: Efficiency of using nonbinary LDPC codes and HD-Cascade for different QBER values for $q = 8$. Bottom: Number of decoding tries in the blind scheme used for the LDPC codes, i.e., the number of messages required. The number of messages required for HD-Cascade can be seen in Fig. 5



4 Discussion

4.1 Nonbinary LDPC codes

Nonbinary LDPC codes are a natural candidate for the information reconciliation stage of HD-QKD, as their order can be matched to the dimension of the used qudits, and they are known to have good decoding performance [9]. Although they typically come with increased decoding complexity, this drawback is less of a concern in this context, since the keys can be processed and stored before being employed in real-time applications, which reduces the significance of decoding latency. Nevertheless, less complex decoder algorithms like EMS [53] or TEMS [54] can be considered to allow

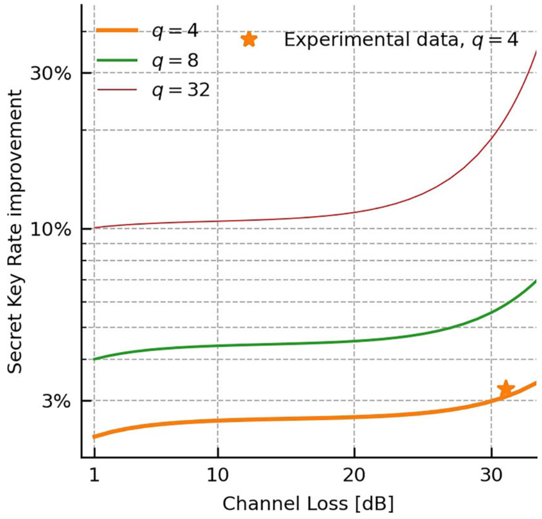


Fig. 7 Relative improvement of the secret key rate for using HD-Cascade compared to binary Cascade. Experimental data provided by recent experiment [52]

the usage of longer codes and for increasing the throughput with a small penalty on efficiency.

The node degree distributions we constructed show ensemble efficiencies close to one, $1.037 - 1.067$ for $q = 4$ and $1.024 - 1.080$ for $q = 8$. To the best of our knowledge, there is no inherent reason for the efficiencies of $q = 8$ to be lower than for $q = 4$. It is rather just a heuristic result due to optimization parameters fitting better. Although the ensembles we found display thresholds near the Slepian–Wolf bound, we believe that even better results could be achieved by expanding the search of the hyperparameters involved in the optimization, such as the enforced sparsity and the highest degree of λ, and by performing a finer sweep of the QBER during density evolution. The evaluated efficiency of finite-size codes shows them performing significantly worse than the thresholds computed with density evolution, with efficiencies ranging from 1.078 to 1.14 for QBER values in a medium range. This gap can be reduced by using longer codes and improving the code construction, e.g., using improved versions of the PEG algorithm [55, 56]. The dependency of the efficiency on the QBER can further be reduced, i.e., flatting the curve in Fig. 6, by improving the position of punctured bits [57].

While working on this manuscript, the usage of nonbinary LDPC codes for information reconciliation has also been proposed in [58]. They suggest mapping symbols of high dimensionality to symbols of lower dimensionality but still higher than 2 if beneficial, in similarity to the layered scheme. This can further be used to decrease computational complexity if required.

4.2 HD-cascade

HD-Cascade has improved performance on high-dimensional QKD setups compared to directly applying binary Cascade. We can see significant improvement in efficiency, with mean efficiencies of $f_{\text{HD-Cascade}} = 1.06, 1.07, 1.12$ compared to $f_{\text{Cascade}} = 1.22,$

1.36, 1.65 for $q = 4, 8, 32$, respectively. Using the setup parameters of a recent experimental implementation of four-dimensional QKD [52], a resulting improvement in range and secret key rate can be observed, especially for higher dimensions. For $q = 32$, an increase of more than 10% in secret key rate over all QBER values and an additional 2.5 dB in tolerable channel loss are achievable according to our simulation results.

The serial approach demonstrates high efficiency across all QBER values but suffers from a strong increase in execution time with higher error rates and an impractical requirement on communication rounds. Apart from the inherent scaling of Cascade with the QBER that is also present for binary implementations, this is additionally attributable to the immediate cascading of partner bits. This penalty can be greatly reduced when implementing the parallel high-dimensional version. The resulting penalty on the mean efficiency is very small with mean efficiencies of $f_{\text{HD-Cascade, practical}} = 1.07, 1.08$ for $q = 4, 16$, respectively. We again see the sawtooth pattern that can also be observed in binary Cascade and is attributed to discrete jumps in block sizes [47]. We observed a significantly higher variance in the efficiency with respect to QBER for the parallel approach, especially for $q = 4$. We assume that this is due to better-performing block size selection for $q = 16$ and that an optimized parameter selection will reduce the sawtooth pattern and increase efficiency overall for all values of q. The search for optimized parameters has been investigated and is with great impact on the efficiency of binary Cascade, see again [47] for an overview. The search for well-performing block size selections for HD-Cascade could therefore be an interesting object of further research.

While the many rounds of communication required by Cascade have raised concerns about resulting limitations on throughput, recent research has shown that sophisticated software implementations can enable Cascade to achieve high throughput even with realistic latency on the classical channel [14, 15]. While high-dimensional Cascade has little additional computations compared to binary Cascade, the number of messages is increased due to the additional rounds of cascading on the partner bits. For $q = 4$, 16, the number of messages per frame can be seen in Fig. 5. Notably, the number of messages required seems to decrease with increasing dimension, resulting in an increasing throughput with increasing dimension. The impact on throughput is dependent on the block size, communication delay, and computing time, an analysis of this for the binary case can be found in [15] and is required for the high-dimensional case before a general behavior can be attributed. This is in contrast to LDPC codes whose throughput would scale negatively with increasing dimension. In our simulations, the relative increase in throughput between $q = 4$ and $q = 16$ is 30% for a QBER of 1% and a delay of 1 ms between Alice and Bob.

4.3 Comparison

Overall, HD-Cascade and nonbinary LDPC codes show good efficiency over all relevant QBER values, with HD-Cascade performing slightly better in terms of efficiency (see Fig. 6) at the expense of increased interactivity. Both show significant improvement compared to the layered scheme. The performance of the layered scheme can be seen for $q = 32$ in Fig. 4, notably for a much smaller block length (data read off Fig. 5 in [8]). Later experimental implementations report efficiencies of $f = 1.25$ [59] ($q = 3$, $n = 1944$, $p = 8\%$) and $f = 1.17$ [60] ($q = 1024$, $n = 4000$, $p = 39.6\%$). These papers report their efficiencies in the β-notation, we converted them to the f-notation for comparison. β is commonly used in the continuous-variable QKD community, whereas f is more widespread with respect to discrete-variable QKD. They can be related via [47]

$$\beta(\mathrm{H}(X) - \mathrm{H}(X|Y)) = \mathrm{H}(X) - f\mathrm{H}(X|Y). \tag{17}$$

HD-Cascade shows a flat efficiency behavior over all ranges, compared to the LDPC codes, which have a bad performance for very low QBER values and show an increase in performance with increasing QBER, see Table 1. This behavior can also be observed in LDPC codes used in binary QKD [17, 18, 21]. While the focus of this work lies in introducing new methods for high-dimensional information reconciliation with good efficiencies, the throughput is another important measure, especially with continuously improving input rates from advancing QKD hardware implementation. While an absolute and direct comparison of throughput strongly depends on the specific implementation and setup parameters and is not a contribution of this work, relative performances can be considered. Cascade has low computational complexity but high interactivity which can limit throughput in scenarios where the classical channel has a high latency.

Nonbinary LDPC codes, on the other hand, have low requirements on interactivity (usually below 10 syndromes per frame using the blind scheme, see Fig. 6) but high computational costs at the decoder. Their decoding complexity scales strongly with q but only slightly with the QBER, as its main dependence is on the number of entries in its parity check matrix and the node degrees. It should be noted that the QBER is usually fairly stable until the loss approaches the maximum range of the setup, e.g., see Fig. 4, and that higher dimensions tend to operate at higher QBER values [7]. It should be emphasized that for QKD, low latency in post-processing is often not a priority as keys do not need to be available immediately but can be stored for usage, with varying importance for different network scenarios. QKD systems can be significantly bigger and more expensive than setups for classical communication. This allows for reconciliation schemes with comparatively high latency and high computational complexity, for example, by extensive usage of pipelining [14, 61, 62].

5 Conclusion

We introduced two new methods for the information reconciliation stage of high-dimensional quantum key distribution and investigated their performance. We present nonbinary LDPC codes designed and optimized specifically for the use in high-dimensional QKD systems. They allow for reconciliation with good efficiency while maintaining a low interactivity between the two parties. For HD-QKD systems of dimension 8, the codes we constructed show efficiencies between $f = 1.078$ and $f = 1.14$ for QBER values between 3% and 15%. We further propose new modifications to Cascade to make it suitable for high-dimensional QKD that greatly increase its efficiency on those systems. The main modification manifests in the request of partner bits as additional means of detecting errors in Cascade. Our simulations show mean efficiencies of $f = 1.06$, 1.07 and 1.12 for dimensions 4, 8 and 32. We also analyze the impact of HD-Cascade on the secret key rate compared to using binary Cascade and note significant improvement, i.e., more than 10% for a 32-dimensional system for all possible channel losses and an increase in range corresponding to an additional 2.5 dB loss.

Appendix A: workflow of HD-cascade

HD-Cascade largely follows the description of binary Cascade in [22] with the modifications described in Sect. 2.3.2 and Sect. 2.3.3. We give here a detailed description of the workflow of HD-Cascade for Bob's side, focusing on the more practical parallel implementation described in Sect. 2.3.3.

Preliminaries: Two q-ary strings of length n, denoted by $\mathbf{x}$ and $\mathbf{y}$, and an estimate of the qudit error rate, denoted by QBER_{SYM}, in slight abuse of notation.

A.1 General flow

HD-Cascade, in general, follows an iteration pattern where for each iteration a permutation is applied. The key, i.e., the binary string, is then divided into blocks and a binary search locates errors. Each iteration is then followed by a Cascade step.

1. *Mapping* Initially, all symbols are mapped to an appropriate binary representation. Prior to the first iteration, all bits are shuffled while maintaining a record of which bits originate from the same symbol. These bits are denoted as ”partner bits". This mapping effectively reduces the expected QBER used for block size calculations in all iterations to $\text{QBER}_{\text{BIN}} = \frac{q}{2(q-1)}\text{QBER}_{\text{SYM}}$. A storage container for keeping track of the smallest block and its parity value each bit takes part in for each iteration is initialized.
2. *First iteration* In the first iteration, the binary representation is split into $v = \log_2(q)$ groups of size n; n being the number of qudits per frame. Each group only contains bits of the same bit-plane, i.e., the first group contains all the first bits of all symbols, the second group contains all the second bits of all symbols and so on. Permutations that are restricted to each group only are applied, i.e., we shuffle only inside each bit-plane. For the first group, the following is done:

   (a) Divide the bits of the group into top-level blocks of size k_1,

   $$k_1 = \min(2^{[\log_2(1/\text{QBER}_{\text{BIN}})]}, n/2) \tag{18}$$

   Calculate and compare the parities of these top-level blocks between Alice and Bob. Store the top-level blocks as the smallest block in iteration 1 for all bits inside blocks with matching parity. On blocks with mismatching parity, a

binary search is executed to locate an error.
Binary Search
(i) Split the respective block in half.
(ii) Calculate and exchange the parities of the two sub-blocks. One of the sub-blocks has a mismatching parity. Store the matching block as the smallest block for its bits in iteration 1.
(iii) If the mismatching sub-block contains only 1 bit, the error is found. Otherwise, repeat Step (a) with the mismatching sub-block.
After locating the error positions of one group using binary search, the corresponding partner bits can be located in all other groups. Request Alice to directly send the values of all partner bits and correct them. This reduces the expected QBER for all other groups. When calculating the top-level block sizes of the following blocks, adjust the expected QBER using the number of partner bits. It can be calculated for group i, $i = 1, ..., v$ as:

$$\mathrm{QBER}_i = \mathrm{QBER}_{\mathrm{BIN}} - \frac{1}{2n} \sum_{j<i} \frac{\mathrm{PB}_j}{v}, \tag{19}$$

where PB_j denotes the number of partner bit requests originating from group j. Repeat step (a) for all other groups but replace $\mathrm{QBER}_{\mathrm{BIN}}$ with QBER_i. Collect all partner bits that are part of a group that has already been processed, e.g., some of the partner bits that occur when handling group $i = 2$ but are part of group $i = 1$. After completing all groups, use those partner bits as input for the Cascade step of the first iteration if their value differs between Bob and Alice. The Cascade step is described separately in Sect. 1

3. *Second iteration* The binary frame is again split into groups; this is also done in the binary protocol [22]. The grouping follows the size t of the smallest block each bit was part of in iteration 1, e.g., all bits whose top-level blocks matched are in one group, then all bits that were part of a block of half the top-level size that had matching parities between Alice and Bob, and so on. The QBER is adjusted according to the block size t as [22]:

$$\mathrm{QBER}_{\mathrm{2nd}}(t) = \mathrm{QBER}_{\mathrm{BIN}} \frac{p_{\mathrm{odd}}(t-1, \mathrm{QBER}_{\mathrm{BIN}})}{p_{\mathrm{even}}(t, \mathrm{QBER}_{\mathrm{BIN}})} \tag{20}$$

$$p_{\mathrm{odd}}(t, p) = \frac{1-(1-2p)^t}{2} \tag{21}$$

$$p_{\mathrm{even}}(t, p) = \frac{1+(1-2p)^t}{2} \tag{22}$$

The top-level block size for each group is then calculated as

$$k_2 = \min(2^{[\log_2(2q/\mathrm{QBER}_{\mathrm{2nd}})]}, n_j/2), \tag{23}$$

with n_j denoting the number of bits in group j. Proceed for each group as described for the first iteration but additionally keep track of all positions where an error is found by the binary search directly. Pass all relevant partner bits and error positions to the Cascade step.

4. *Remaining iterations* The remaining iterations follow a more basic pattern. First, apply a random permutation to the binary frame. Divide the frame into top-level blocks of size k_i. As is common in binary Cascade, a doubling of the block sizes is done for each iteration, i.e., $k_{i+1} = 2k_i$, starting with $k_3 = n/16$. The impact of the remaining iterations on the efficiency is rather low, as most errors can be expected to be corrected after the second iteration. Proceed for all blocks as described for iteration 1 and 2. Feed all known error positions to the Cascade step.

A.2 The cascade step

The Cascade step expects a list of error positions as input. All those errors are already corrected, i.e., their bit value is flipped in Bob's current version of the key.

1. Go over all error positions and find the smallest block they take part in from all iterations that have not yet been processed. The term iteration refers to the iterations as described in Sect. 1. Note that each iteration applies a respective permutation. Pass the blocks to the next step.
   Example: *Consider the case where the bit at position 865 is incorrect. We check the smallest block information for this bit and see that it took part in a block of size 16, 8, and 4 in iterations 1,2, and 3, respectively. Iterations 2 and 3 are already flagged as processed for this bit, so proceed to the next step with the block of size 16 from iteration 1.*
2. For all blocks selected in Step 1, recalculate Bob's parity. Alice's parity is still known. If there is a mismatch, try to locate a new error by running a binary search on the blocks. This can be done either in parallel or in series. If done in parallel, multiple blocks originating from different iterations might point to the same error. Correct all found errors and request the respective partner bits from Alice. During the binary search new, smaller blocks are created. Update the smallest block information for all respective bits and iterations. Append all newly found errors, i.e., from both binary search and requested partner bits, to the list of known error positions. Mark the respective iterations as processed.
   Example: *We recalculate the parity of the block with size 16 from the previous step. Its parity now mismatches with that of Alice, and we perform a binary search on the block. As a result, a new error is found at position 35475. In the process, new blocks of size 8,4,2, and 1 with matching parity are created for iteration 1. The smallest block information is updated accordingly. We request the partner bits and receive the values of bits 35476 and 35477 of which only 35476 differs between Alice and Bob. Bit positions 35475 and 35476 are appended to the list of found errors. Iteration 1 is flagged as processed for bits 865 and 35475.*
3. Go over all error positions and check for each bit if all the smallest blocks it takes part in are already processed. If so, remove that position from the list. If the list of all error positions is empty, end the Cascade step, otherwise, repeat all steps.
   Example: *All iterations are now flagged for bit 865 and it is therefore removed from the list of errors.*

Acknowledgements The Center of Excellence SPOC (ref DNRF123).

Author Contributions R.M. developed and implemented the information reconciliation algorithms and wrote the main manuscript text. D.R. and M.Z provided the data and code to conduct the secret key rate calculations. All authors reviewed the manuscript.

Funding Open access funding provided by Technical University of Denmark.

Data availability All data used in this work are available from the corresponding author upon reasonable request.

Declarations

Conflict of interest The authors declare no competing financial or non-financial interests.

References

1. Bennett, C.H., Brassard, G.: Quantum cryptography: Public key distribution and coin tossing. In: Proceedings of IEEE International Conference on Computers, Systems, and Signal Processing, India, p. 175 (1984)
2. Pirandola, S., Andersen, U.L., Banchi, L., Berta, M., Bunandar, D., Colbeck, R., Englund, D., Gehring, T., Lupo, C., Ottaviani, C., Pereira, J.L., Razavi, M., Shaari, J.S., Tomamichel, M., Usenko, V.C., Vallone, G., Villoresi, P., Wallden, P.: Advances in quantum cryptography. Advances in Optics and Photonics **12**(4), 1012 (2020). https://doi.org/10.1364/aop.361502
3. Cozzolino, D., Da Lio, B., Bacco, D., Oxenlø we, L.K.: High-dimensional quantum communication: benefits, progress, and future challenges. Adv. Quantum Technol. **2**(12), 1900038 (2019). https://doi.org/10.1002/qute.201900038
4. Bechmann-Pasquinucci, H., Tittel, W.: Quantum cryptography using larger alphabets. Phys. Rev. A **61**, 062308 (2000). https://doi.org/10.1103/PhysRevA.61.062308
5. Niu, M.Y., Xu, F., Shapiro, J.H., Furrer, F.: Finite-key analysis for time-energy high-dimensional quantum key distribution. Phys. Rev. A **94**, 052323 (2016). https://doi.org/10.1103/PhysRevA.94.052323
6. Ecker, S., Bouchard, F., Bulla, L., Brandt, F., Kohout, O., Steinlechner, F., Fickler, R., Malik, M., Guryanova, Y., Ursin, R., Huber, M.: Overcoming noise in entanglement distribution. Phys. Rev. X **9**, 041042 (2019). https://doi.org/10.1103/PhysRevX.9.041042
7. Cozzolino, D., Lio, B.D., Bacco, D., Oxenløwe, L.K.: High-dimensional quantum communication: benefits, progress, and future challenges. Adv. Quantum Technol. **2**(12), 1900038 (2019). https://doi.org/10.1002/qute.201900038
8. Zhou, H., Wang, L., Wornell, G.: Layered schemes for large-alphabet secret keydistribution. In: 2013 Information Theory and Applications Workshop (ITA), pp.1–10 (2013).https://doi.org/10.1109/ITA.2013.6502993
9. Davey, M.C., MacKay, D.J.C.: Low density parity check codes over GF(q). In: 1998 Information Theory Workshop (Cat. No.98EX131), pp. 70–71 (1998). doi:10.1109/ITW.1998.706440
10. Brassard, G., Salvail, L.: Secret-key reconciliation by public discussion. In: Helleseth, T. (ed.) Advances in Cryptology – EUROCRYPT '93, pp. 410–423. Springer, Berlin, Heidelberg (1994)

11. Martinez-Mateo, J., Elkouss, D., Martin, V.: Blind reconciliation. Quantum Info. Comput. **12**(9–10), 791–812 (2012)
12. Kiktenko, E.O., Malyshev, A.O., Fedorov, A.K.: Blind information reconciliation with polar codes for quantum key distribution. IEEE Commun. Lett. **25**(1), 79–83 (2021). https://doi.org/10.1109/lcomm.2020.3021142
13. Martinez-Mateo, J., Elkouss, D., Martin, V.: Key reconciliation for high performance quantum key distribution. Sci. Rep. **3**(1), 1576 (2013). https://doi.org/10.1038/srep01576
14. Mao, H.-K., Li, Q., Hao, P.-L., Abd-El-Atty, B., Iliyasu, A.M.: High performance reconciliation for practical quantum key distribution systems. Opt. Quant. Electron. **54**(3), 163 (2022). https://doi.org/10.1007/s11082-021-03489-4
15. Pedersen, T.B., Toyran, M.: High performance information reconciliation for qkd with cascade. Quantum Info. Comput. **15**(5–6), 419–434 (2015)
16. Van Assche, G., Cardinal, J., Cerf, N.J.: Reconciliation of a quantum-distributed gaussian key. IEEE Trans. Inf. Theory **50**(2), 394–400 (2004). https://doi.org/10.1109/TIT.2003.822618
17. Mao, H., Li, Q., Han, Q., Guo, H.: High-throughput and low-cost LDPC reconciliation for quantum key distribution. Quantum Inf. Process. **18**(7), 232 (2019). https://doi.org/10.1007/s11128-019-2342-2
18. Elkouss, D., Leverrier, A., Alleaume, R., Boutros, J.J.: Efficient reconciliation protocol for discrete-variable quantum key distribution. In: 2009 IEEE International Symposium on Information Theory, pp. 1879–1883 (2009). doi:10.1109/ISIT.2009.5205475
19. Kiktenko, E.O., Trushechkin, A.S., Lim, C.C.W., Kurochkin, Y.V., Fedorov, A.K.: Symmetric blind information reconciliation for quantum key distribution. Phys. Rev. Appl. **8**, 044017 (2017). https://doi.org/10.1103/PhysRevApplied.8.044017
20. Treeviriyanupab, P., Zhang, C.-M.: Efficient integration of rate-adaptive reconciliation with syndrome-based error estimation and subblock confirmation for quantum key distribution. Entropy (2024). https://doi.org/10.3390/e26010053
21. Kasai, K., Matsumoto, R., Sakaniwa, K.: Information reconciliation for QKD with rate-compatible non-binary LDPC codes. In: 2010 International Symposium On Information Theory and Its Applications, pp. 922–927 (2010). doi:10.1109/ISITA.2010.5649550
22. Pacher, C., Grabenweger, P., Martinez Mateo, J., Martin, V.: An information reconciliation protocol for secret-key agreement with small leakage pp. 730–734 (2015). https://doi.org/10.1109/ISIT.2015.7282551
23. Liveris, A.D., Xiong, Z., Georghiades, C.N.: Compression of binary sources with side information at the decoder using LDPC codes. IEEE Commun. Lett. **6**(10), 440–442 (2002). https://doi.org/10.1109/LCOMM.2002.804244
24. Bennett, C.H., Brassard, G., Robert, J.-M.: Privacy amplification by public discussion. SIAM J. Comput. **17**(2), 210–229 (1988). https://doi.org/10.1137/0217014
25. Lo, H.-K.: Method for decoupling error correction from privacy amplification. New J. Phys. **5**, 36–36 (2003). https://doi.org/10.1088/1367-2630/5/1/336
26. Tupkary, D., Lütkenhaus, N.: Using cascade in quantum key distribution. Phys. Rev. Appl. **20**, 064040 (2023). https://doi.org/10.1103/PhysRevApplied.20.064040
27. Slepian, D., Wolf, J.: Noiseless coding of correlated information sources. IEEE Trans. Inf. Theory **19**(4), 471–480 (1973). https://doi.org/10.1109/TIT.1973.1055037
28. Tomamichel, M., Martinez-Mateo, J., Pacher, C., Elkouss, D.: Fundamental finite key limits for one-way information reconciliation in quantum key distribution. Quantum Inf. Process. (2017). https://doi.org/10.1007/s11128-017-1709-5
29. Lütkenhaus, N.: Estimates for practical quantum cryptography. Phys. Rev. A **59**(5), 3301–3319 (1999). https://doi.org/10.1103/physreva.59.3301
30. Fung, C.-H.F., Ma, X., Chau, H.F.: Practical issues in quantum-key-distribution postprocessing. Phys. Rev. A (2010). https://doi.org/10.1103/physreva.81.012318
31. Boaron, A., Korzh, B., Houlmann, R., Boso, G., Rusca, D., Gray, S., Li, M.-J., Nolan, D., Martin, A., Zbinden, H.: Simple 2.5 GHz time-bin quantum key distribution. Appl. Phys. Lett. **112**(17), 171108 (2018). https://doi.org/10.1063/1.5027030
32. Lin, S., Li, J.: Fundamentals of Classical and Modern Error-Correcting Codes. Cambridge University Press, Cambridge (2021)
33. Li, G., Fair, I., Krzymien, W.: Density evolution for nonbinary ldpc codes under gaussian approximation. Inf. Theory, IEEE Trans. on **55**, 997–1015 (2009). https://doi.org/10.1109/TIT.2008.2011435

34. Aruna, S., Anbuselvi, M.: FFT-SPA based non-binary LDPC decoder for IEEE 802.11n standard. In: 2013 International Conference on Communication and Signal Processing, pp. 566–569 (2013). https://doi.org/10.1109/iccsp.2013.6577118
35. Wymeersch, H., Steendam, H., Moeneclaey, M.: Log-domain decoding of LDPCcodes over GF(q). In: 2004 IEEE International Conference on Communications (IEEE Cat. No.04CH37577), vol. 2, pp. 772–7762 (2004). https://doi.org/10.1109/ICC.2004.1312606
36. Li, G., Fair, I.J., Krzymien, W.A.: Density evolution for nonbinary LDPC codes under gaussian approximation. IEEE Trans. Inf. Theory **55**(3), 997–1015 (2009). https://doi.org/10.1109/TIT.2008.2011435
37. Dupraz, E., Savin, V., Kieffer, M.: Density evolution for the design of non-binary low density parity check codes for Slepian-Wolf coding. IEEE Trans. Commun. **63**(1), 25–36 (2015). https://doi.org/10.1109/TCOMM.2014.2382126
38. Elkouss, D., Martinez-Mateo, J., Martin, V.: Secure rate-adaptive reconciliation. In: 2010 International Symposium On Information Theory and Its Applications. IEEE, (2010). https://doi.org/10.1109/isita.2010.5650099
39. Elkouss, D., Martínez-Mateo, J., Martín, V.: Secure rate-adaptive reconciliation. In: 2010 International Symposium On Information Theory and Its Applications, pp. 179–184 (2010). https://doi.org/10.1109/ISITA.2010.5650099
40. Kiktenko, E.O., Trushechkin, A.S., Lim, C.C.W., Kurochkin, Y.V., Fedorov, A.K.: Symmetric blind information reconciliation for quantum key distribution. Phys. Rev. Appl. **8**, 044017 (2017). https://doi.org/10.1103/PhysRevApplied.8.044017
41. Savin, V.: Non binary LDPC codes over the binary erasure channel: Density evolution analysis. In: 2008 First International Symposium on Applied Sciences on Biomedical and Communication Technologies, pp. 1–5 (2008). https://doi.org/10.1109/ISABEL.2008.4712621
42. Richardson, T.J., Urbanke, R.L.: The capacity of low-density parity-check codes under message-passing decoding. IEEE Trans. Inf. Theory **47**(2), 599–618 (2001). https://doi.org/10.1109/18.910577
43. Gorgoglione, M., Savin, V., Declercq, D.: Optimized puncturing distributions for irregular non-binary LDPC codes. In: 2010 International Symposium On Information Theory & Its Applications, pp. 400–405 (2010). https://doi.org/10.1109/ISITA.2010.5649264
44. Chung, S.-Y., Richardson, T.J., Urbanke, R.L.: Analysis of sum-product decoding of low-density parity-check codes using a gaussian approximation. IEEE Trans. Inf. Theory **47**(2), 657–670 (2001). https://doi.org/10.1109/18.910580
45. Storn, R., Price, K.: Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. J. Global Optim. **11**, 341–359 (1997). https://doi.org/10.1023/A:1008202821328
46. Buttler, W.T., Lamoreaux, S.K., Torgerson, J.R., Nickel, G.H., Donahue, C.H., Peterson, C.G.: Fast, efficient error reconciliation for quantum cryptography. Phys. Rev. A **67**, 052303 (2003). https://doi.org/10.1103/PhysRevA.67.052303
47. Martínez Mateo, J., Pacher, C., Peev, M., Ciurana, A., Martin, V.: Demystifying the information reconciliation protocol cascade. Quantum Inf. Comput. (2014). https://doi.org/10.26421/QIC15.5-6-6
48. Hu, X.-M., Zhang, C., Guo, Y., Wang, F.-X., Xing, W.-B., Huang, C.-X., Liu, B.-H., Huang, Y.-F., Li, C.-F., Guo, G.-C., Gao, X., Pivoluska, M., Huber, M.: Pathways for entanglement-based quantum communication in the face of high noise. Phys. Rev. Lett. **127**, 110505 (2021). https://doi.org/10.1103/PhysRevLett.127.110505
49. Hu, X.-Y., Eleftheriou, E., Arnold, D.M.: Regular and irregular progressive edge-growth tanner graphs. IEEE Trans. Inf. Theory **51**(1), 386–398 (2005). https://doi.org/10.1109/TIT.2004.839541
50. Rusca, D., Boaron, A., Grünenfelder, F., Martin, A., Zbinden, H.: Finite-key analysis for the 1-decoy state QKD protocol. Appl. Phys. Lett. **112**(17), 171104 (2018). https://doi.org/10.1063/1.5023340
51. Vagniluca, I., Lio, B.D., Rusca, D., Cozzolino, D., Ding, Y., Zbinden, H., Zavatta, A., Oxenløwe, L.K., Bacco, D.: Efficient time-bin encoding for practical high-dimensional quantum key distribution. Phys. Rev. Appl. (2020). https://doi.org/10.1103/physrevapplied.14.014051
52. Zahidy, M., Ribezzo, D., De Lazzari, C., Vagniluca, I., Biagi, N., Occhipinti, T., Oxenløwe, L.K., Galili, M., Hayashi, T., Antonelli, C., Mecozzi, A., Zavatta, A., Bacco, D.: 4-dimensional quantum key distribution protocol over 52-km deployed multicore fibre. In: 2022 European Conference on Optical Communication (ECOC), pp. 1–4 (2022)
53. Declercq, D., Fossorier, M.: Decoding algorithms for nonbinary LDPC codes over GF(q). IEEE Trans. Commun. **55**(4), 633–643 (2007). https://doi.org/10.1109/TCOMM.2007.894088

54. Li, E., Gunnam, K., Declercq, D.: Trellis based extended Min-Sum for decoding nonbinary LDPC codes. In: 2011 8th International Symposium on Wireless Communication Systems, pp. 46–50 (2011). https://doi.org/10.1109/ISWCS.2011.6125307
55. Xiao, H., Banihashemi, A.H.: Improved progressive-edge-growth (PEG) construction of irregular LDPC codes. IEEE Commun. Lett. **8**(12), 715–717 (2004). https://doi.org/10.1109/LCOMM.2004.839612
56. Venkiah, A., Declercq, D., Poulliat, C.: Randomized progressive edge-growth (RandPEG). In: 2008 5th International Symposium on Turbo Codes and Related Topics, pp. 283–287 (2008). https://doi.org/10.1109/TURBOCODING.2008.4658712
57. Gorgoglione, M., Savin, V., Declercq, D.: Optimized puncturing distributions for irregular non-binary LDPC codes. In: 2010 International Symposium On Information Theory and Its Applications, pp. 400–405 (2010). https://doi.org/10.1109/ISITA.2010.5649264
58. Mitra, D., Tauz, L., Sarihan, M.C., Wong, C.W., Dolecek, L.: Non-Binary LDPC Code Design for Energy-Time Entanglement Quantum Key Distribution. Arxive (2023)
59. Liu, J.-Y., Lin, Z., Liu, D., Feng, X., Liu, F., Cui, K., Huang, Y., Zhang, W.: High-dimensional quantum key distribution using energy-time entanglement over 242 km partially deployed fiber. Quantum Science and Technology (2022). https://doi.org/10.1088/2058-9565/acfe37
60. Zhong, T., Zhou, H., Horansky, R.D., Lee, C., Verma, V.B., Lita, A.E., Restelli, A., Bienfang, J.C., Mirin, R.P., Gerrits, T., Nam, S.W., Marsili, F., Shaw, M.D., Zhang, Z., Wang, L., Englund, D., Wornell, G.W., Shapiro, J.H., Wong, F.N.C.: Photon-efficient quantum key distribution using time-energy entanglement with high-dimensional encoding. New J. Phys. **17**(2), 022002 (2015). https://doi.org/10.1088/1367-2630/17/2/022002
61. Yang, S.-S., Liu, J.-Q., Lu, Z.-G., Bai, Z.-L., Wang, X.-Y., Li, Y.-M.: An FPGA-based LDPC decoder with ultra-long codes for continuous-variable quantum key distribution. IEEE Access **9**, 47687–47697 (2021)
62. Cui, K., Wang, J., Zhang, H.-F., Luo, C.-L., Jin, G., Chen, T.-Y.: A real-time design based on FPGA for expeditious error reconciliation in QKD system. IEEE Trans. Inf. Forensics Secur. **8**(1), 184–190 (2013). https://doi.org/10.1109/TIFS.2012.2228855