IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 17 July 2023, accepted 5 August 2023, date of publication 7 August 2023, date of current version 15 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3303392

EXPOSITION

Cracking the Curious Case of the Cascade Protocol

ANAND CHOUDHARY, (Student Member, IEEE), AND AJAY WASAN
Department of Physics, Indian Institute of Technology Roorkee, Roorkee, Uttarakhand 247667, India

Corresponding author: Anand Choudhary (anand_c@ph.iitr.ac.in)

This work was supported in part by the Department of Science and Technology (DST), Government of India, through the Quantum Enabled Science and Technology (QuEST) Program; and in part by the I-HUB Quantum Technology Foundation through the National Mission on Interdisciplinary Cyber-Physical Systems (NM-ICPS).

ABSTRACT With the nature-powered quantum revolution gaining momentum, Quantum Computation and Quantum Information are all set to transform the landscape of all areas of science and technology. Quantum Cryptography is one such field which has witnessed tremendous progress in theory as well as in its practical implementation. Quantum Key Distribution is one of its integral parts and it necessarily requires the usage of an information reconciliation protocol to correct the errors present in the key bits transmitted via the physical system involved. Cascade is one such protocol which enjoys popularity among such system developers due to its simplicity and high error-correction efficiency. In this work, we identify, improve, interpret, and implement key optimizations from a practical viewpoint and give an algorithmic implementation of the Cascade protocol which would serve to increase its overall efficiency without changing its foundational structure. We also dig deep into the math and physical interpretation behind the values and expressions of key parameters and quantities of Cascade, which effectively make use of the cascade of errors unearthed by it so as to not only ensure an exponentially decreasing probability of any errors in the key bits with the passes of Cascade but also ensure that only the minimum possible information is leaked to a potential eavesdropper. Finally, we also give a visual walk-through of Cascade in action and support it with explanations at every step. By doing so, we aim to equip researchers and developers with an insight driven error correction tool which is sure to be useful in practice.

INDEX TERMS Cascade protocol, error correction, information reconciliation, quantum communication, quantum cryptography, quantum key distribution (QKD).

I. INTRODUCTION AND RELATED WORK

At the most fundamental level, the magical Nature runs on the mysterious yet mind-blowing laws of quantum mechanics. Harnessing this power of Nature by thinking *naturally* about computation and information processing has led to the birth of an entirely rich, new paradigm: Quantum Computation and Quantum Information (QCQI), which has started to unlock exciting new possibilities in all spheres of science and technology such as in cryptography, system simulation, drug discovery, healthcare, finance, renewable energy, artificial intelligence, etc. [1]

The associate editor coordinating the review of this manuscript and approving it for publication was Derek Abbott.

Though many of the applications of QCQI in the aforementioned areas are in a nascent stage, Quantum Cryptography is one of its sub-fields that has not only seen numerous advances in theory but has also seen them being successfully implemented in practice [2], [3], [4], [5]. In general, the research in cryptography revolves around developing techniques or protocols so as to ensure secure communication or computation between two or more parties in the presence of any adversary. Obviously, transmission of secure messages inevitably requires encryption and/or decryption keys. The hitherto ubiquitous *classical* methods of key distribution based on public key cryptography, such as the DH [6] and RSA [7] protocols, can be easily broken by Shor's quantum algorithm for factoring [8], thus, posing a huge security risk

worldwide! Further, if we turn to private key cryptography instead, a major disadvantage of the classical protocols concerned is that any eavesdropper (Eve), the adversary, can easily listen in to the secret key being transmitted, say, between the sender (Alice) and the receiver (Bob) without any risk of being detected by either of them. Quantum Key Distribution (QKD) resolves these problems by exploiting the quantum mechanical principle that the measurement of a system, in general, disturbs the state of the system. Consequently, via eavesdropping, Eve cannot possibly gain any amount of information about the secret key being transmitted from Alice to Bob without measuring it in some way or the other. While doing so, Eve would undeniably introduce some alterations in the key, which can then be easily detected by Alice and Bob. The QKD protocol can thus be aborted by Alice and Bob if these alterations exceed a set threshold and they can start all over again. Clearly, the security of QKD is guaranteed by the principles of quantum mechanics and thus, it remains risk-free given the validity of the fundamental laws of Nature [9]!

The distribution of the secret key bits between Alice and Bob is inevitably prone to error even in the absence of any eavesdropping, since no physical communication system can possibly be completely error-free. Consequently, it is extremely important for Alice and Bob to reconcile the correlated information possessed by them in the form of their *sifted* key strings and thereby, correct errors in them so as to obtain as similar a key as possible. This is accomplished by means of an information reconciliation, essentially, an error-correction protocol [1]. We emphasized on the word 'sifted' because Alice and Bob sift through the bits of their respective key strings and only retain those bits in which Bob's state measurement basis coincides with Alice's state preparation basis; otherwise, even in an ideal, error-free setting, the *raw* bit values possessed by Alice and Bob can possibly be different.

The Cascade protocol, which was proposed by Brassard and Salvail in their seminal paper in 1993 [10], is the most widely used protocol for error correction in QKD systems worldwide. Despite the high time overhead associated with its usage and in spite of the fact that numerous other error correction protocols have since been proposed [11], [12], [13], in practice, none of them have become as popular as the original Cascade protocol. This is quite likely due to its simplicity and ease in implementation coupled with the fact that it has a high level of error correction efficiency [17].

Although many new optimizations to the Cascade protocol have subsequently been proposed, only a few of those propositions optimize the Cascade protocol and/or alter its choice of parameters without modifying the protocol's foundational structure [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [27]. Only such studies have been considered by us for the purpose of comparison.

A previous study [17] has analysed numerous such optimizations and has put forth some new sets of optimizations which have been shown by computer simulations to be outperforming the original Cascade protocol. However, a major problem associated with their proposed optimizations is that they significantly increase the number of interactions taking place between Alice and Bob over a classical channel so as to achieve a better efficiency than the original version of Cascade. This follows from the fact that the number of passes in their optimized versions of Cascade (14 - 16) is extremely high as compared to that in the original version (only 4). In fact, even those sets of optimizations (numbered as 3 and 4 in their study) which have been proposed by them for a comparatively (as compared to the other sets of optimizations proposed by them) reduced time overhead (referred to by them as a communication cost) involve about 30 more channel uses (on an average) than the original version of Cascade (This can be clearly seen in Fig. 9 of their study). In actuality, this average number of extra channel uses is at least 60 ($=2 \times 30$) since their definition of the number of channel uses is, *inter alia*, based on information leakage whereas in reality, irrespective of whether 2 bits of information exchanged between Alice and Bob amount to an information leakage of 1 bit or not, 2 bits of information are physically exchanged via the classical channel, and hence, the channel has been used twice in actuality. Considering the fact that Cascade is already highly interactive, these significantly high *extra* number of interactions in their proposed optimizations can prove to be a severe bottleneck in achieving high key generation rates in practical QKD systems. Thus, it is undesirable to use these and other more recent sets of proposed optimizations that involve a large number of passes of the protocol [22], [23] in practice. The authors of [17] have also published a follow up study [18] wherein, *inter alia*, they have proposed to store the (previously erroneous) bits corrected at every step of the protocol and use their parities, *if possible*, to compute the parities of new blocks. The same optimization has also been suggested in [19]. Moreover, another study has also proposed a very similar optimization involving prefix parity lists [20]. Building up on this, a recent study [21] proposed an optimization termed as 'Block Parity Inference' which involves the storage of all the blocks and subblocks formed during the passes of the Cascade protocol so as to compute the parity of a new block, *if possible*, by only using previously computed parities or linear combinations thereof. However, as is evident, all these aforementioned optimizations impose a significantly high storage and computation overhead as compared to the original Cascade protocol and hence, slow down the error correction process in practice. Therefore, it's not recommended to use them in practical implementations of Cascade and we do not include them in our implementation of Cascade.

Additionally, we found out that, *inter alia*, Optimization 3 of the previously mentioned study [17] sets the value of the initial block size (k_1) in Cascade to $\approx 1/Q$ (here, $Q =$ Quantum Bit Error Rate (QBER) which is the average probability of error in the value of a transmitted qubit.) which is different from the one in the original version of Cascade, i.e.,

$k_1 \approx 0.73/Q$ [10], [24] (the reason for this originally proposed value shall be justified in detail later on in this paper). Quite interestingly, we found out that this study quotes this to be in line with what has been suggested in [16], a study in which it has been stated that the mathematical basis of $k_1 \approx 0.73/Q$ is not yet clear and future work could examine the same. Further on, a common lack of understanding of an upper bound on the information leaked (in terms of the parity bits) in or up to a particular pass of the Cascade protocol is evident in many of these studies which have either stated the same vaguely by not specifying an exact expression for the number of blocks for which a non-zero number of parities would be exchanged in the first place in Cascade [16], [17], [25] (In fact, [16] has again set this aside for future work.) or have considered optimizing an expression for information leakage which is incorrect since it does not take into account the backtracking feature of Cascade – the most important attribute of Cascade that lends it its enhanced error correction ability [22], [23].

It is quite easy to reason that the above lack in understanding is mostly owed to the fact that the original Cascade paper [10] presents these (and other) expressions, and choices for a certain set of parameter values without any detailed explanations. Effectively, the originally proposed Cascade protocol has been rendered as a black box. This fact is also directly echoed by some studies concerning the Cascade protocol which have either effectively quoted the probabilistic analysis/proof (behind assigning only a particular set of values to certain parameters for an exponential reduction in the error probability with the passes of the protocol) verbatim by presenting it with very minor changes [14] or have labelled it as having no theoretical basis at all [22]. To the best of our knowledge, till now, only 1 study [16] is available in the literature which has elaborated (to some extent) on the original proof and pointed out a mistake in it. However, as we show later, this analysis/proof [16] is incomplete, still not completely rigorous, has errors in certain expressions and makes certain adhoc assumptions.

All of this led to the formulation of the two-fold objective of our paper, fulfilling which, here, we not only give a practically useful, optimized algorithmic implementation of Cascade (which doesn't exceed and rather, reduces the number of interactions involved in it without changing its foundational structure.) but also at the same time, extensively clarify the mathematics behind the original protocol and its working mechanism with in-depth physical interpretations and step-by-step illustrations.

II. THE CASCADE PROTOCOL: INTERPRETATION AND IMPLEMENTATION

Cascade [10] is a protocol for information reconciliation, *essentially*, error correction over a shared public channel, which interactively reconciles the errors between the *sifted* key strings P and Q possessed by Alice and Bob (respectively) so that they are ultimately able to obtain a *shared* key string S [1] (In practice, the protocol results in Bob's *sifted* key string being modified to a new key string R such that $R \approx P$).

Let the lengths of the *sifted* binary key strings possessed by Alice and Bob be N. Then, $P = P_0P_1 \dots P_{N-1}$ and $Q = Q_0Q_1 \dots Q_{N-1}$ such that $P, Q \in \mathbb{A}^N$ where $\mathbb{A}$ is the binary alphabet $\{0, 1\}$.

Cascade is an iterative protocol involving several passes, the total number of which, is fixed before its start. It operates on blocks of the *sifted* key strings. Let k_i denote the block size in the i^{th} pass of the protocol. In the 1^{st} pass, Alice and Bob choose a particular value of k_1 (depending on the error probability p [= QBER]) and partition their key strings into blocks of length k_1 bits. The l^{th} block in the 1^{st} pass will consist of the bits whose indices lie in the following set: $K_l^1 = \{m \mid (l-1)k_i \leq m < \min(lk, N)\}$.

Alice then sends the parities of all her blocks (The sum *modulo* 2 of the bits in a block is the parity of a block) to Bob. Algorithm 1 gives an optimized (explained towards the end of this section) algorithmic implementation of how to compute the parity(/parities) of a set(/sets) of indices corresponding to the key bits in a block(/blocks).

Algorithm 1 Parity

```
global N
Function parity (Z_int, key_index_lists) :
    N_l ← |key_index_lists|
    parity_list ← [0] × N_l
    for i ← 0 to N_l − 1 do
        for j ← 0 to |key_index_lists[i]| − 1 do
            key_index_lists[i][j] ← N − 1 − key_index_lists[i][j]
        end
        for k ∈ key_index_lists[i] do
            parity_list[i] ← parity_list[i] ∧ ((Z_int >> k) & 1)
        end
    end
    if |parity_list| = 1 then
        return parity_list[0]
    return parity_list
```

A primitive named BINARY is then used by Bob in order to find and hence, correct *exactly* 1 error in each of those blocks whose parity differs from that of Alice's corresponding block. The BINARY primitive essentially involves an interactive binary search which is to be performed by Alice and Bob so that Bob can find *exactly* 1 error in his bit string. It proceeds in the following way (see Algorithm 2 for an algorithmic implementation):

- Alice sends the parity of the 1^{st} half of the bit string to Bob.
- Bob determines whether there is a parity mismatch (with regards to the corresponding halves in Alice's bit string) in the 1^{st} half or the 2^{nd} half of the string by comparing the parity sent by Alice with the corresponding parity computed by him. Only the half where the parity mismatch has occurred is considered for further iterations.
- These 2 steps are repeatedly applied to the half identified by Bob in the previous step so as to eventually find an error.

Algorithm 2 BINARY

```
global N, P_int, Q_int, net_block_index_lists
Function binary (block_index_list, pass_num) :
    beg ← 0
    end ← N − 1
    while beg < end do
        mid ← ⌊(beg+end)/2⌋
        check_list_1 ← block_index_list[beg ... mid]
        check_list_2 ← block_index_list[mid + 1 ... end]
        Bob sends check_list_1 to Alice
        Alice computes Alice_parity ← parity (P_int, check_list_1)
        Alice sends Alice_parity to Bob
        Bob_parity ← parity (Q_int, check_list_1)
        if Bob_parity ≠ Alice_parity then
            end ← mid
            Append check_list_2 to net_block_index_lists[pass_num]
        else
            beg ← mid + 1
            Append check_list_1 to net_block_index_lists[pass_num]
        end
    end
    bit_flip_index ← block_index_list[beg]
    return bit_flip_index
```

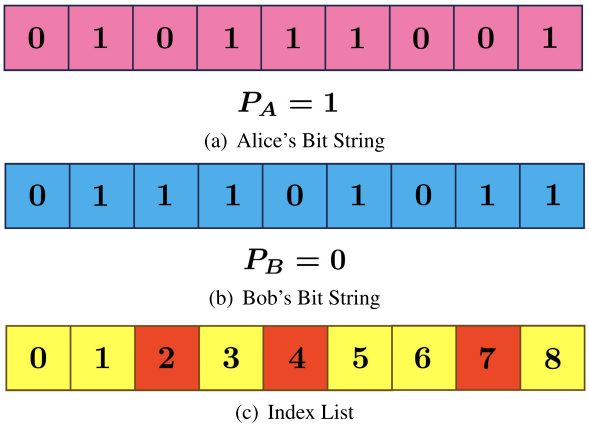


FIGURE 1. Bit strings and the corresponding index list used in Fig. 2 [Error locations are indicated by the colour vermillion; P_A and P_B denote the parity of Alice's and Bob's bit strings respectively].

As an example of the BINARY algorithm in action, a diagrammatic representation of the steps involved in the same is given in Fig. 2. The bit strings (blocks) initially possessed by Alice and Bob are shown in Fig. 1(a) and Fig. 1(b) respectively. As is evident from Fig. 1(c), the bits located at the indices 2, 4 and 7 in Bob's bit string are in error. Since $P_A \neq P_B$ (see Fig. 1)), Bob can find *exactly* 1 error in his bit string via BINARY. As is clear from Fig. 2, application of the BINARY algorithm clearly enables Bob to find the error at index 7 in his bit string via successive comparisons of the parity of the 1st half of his bit string with that of Alice at every step. Note that the errors at indices 2 and 4 could not be detected by BINARY since they together led to a parity match of the corresponding bit string halves in the 1st step of BINARY.

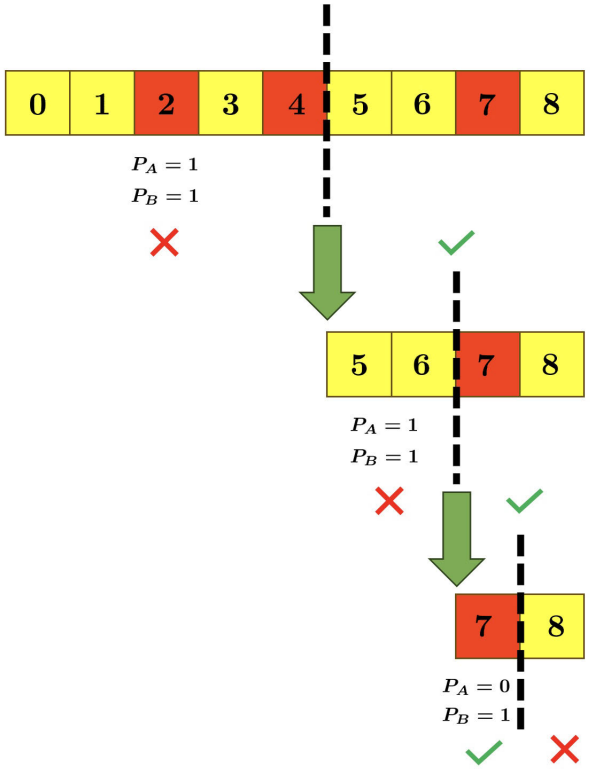


FIGURE 2. Illustration of the steps of the BINARY algorithm as applied to Fig. 1(c) [Error locations are indicated by the colour vermillion; P_A and P_B denote the parity of Alice's and Bob's bit string halves respectively; Parity mismatches between corresponding bit string halves are indicated by green ticks while parity matches are indicated by red crosses].

From the aforementioned example, it is also clear that a parity mismatch between Alice's and Bob's corresponding blocks can only occur when Bob's block has an odd number of errors. Consequently, in such a case, Bob's block would be said to possess an odd error parity. On the other hand, parities of Alice's and Bob's corresponding blocks would be the same when either Bob's block has no errors at all or when it has an even number of errors. Consequently, in such a case, Bob's block would be said to possess an even error parity. Thus, after using BINARY on all the blocks with an odd error parity in the 1st pass of Cascade, all of Bob's blocks would have an even error parity.

For each subsequent pass i (where $i > 1$) of Cascade, Alice and Bob randomly shuffle the bits of their key strings P and Q so that the set of bits that form a block is different from the corresponding set in the previous pass. Doing so clearly opens up the possibility of Bob being able to detect and correct errors which were hidden in the previous pass(es). In order to affect this random shuffling of bits, Alice and Bob both agree on a choice of random permutation function $f_i : [0, \ldots, N-1] \rightarrow [1, \ldots, \lceil \frac{N}{k_i} \rceil]$ (Note that key indexation is from 0 to $N-1$ while block indexation is from 1 to $\lceil \frac{N}{k_i} \rceil$ for the i^{th} pass here). Accordingly, the bits of the key whose indices lie in the following set: $K_l^i = \{m \mid f_i(m) = l\}$ form the l^{th} block in the i^{th} pass.

Akin to the procedure followed in the beginning of the 1st pass, in the i^{th} pass (where $i > 1$), post the shuffling of the bits, Alice sends the parities of all her blocks to Bob. Using BINARY, Bob then finds and corrects exactly one error in each of those blocks which had an odd error parity. However, it's evident that any error that gets corrected in a pass i (where $i > 1$) was hidden (and was hence, not corrected) in the previous pass(es). Accordingly, backtracking and correcting such an error (found in a block in such a pass) in all those blocks formed during the previous pass(es) which contain that error, can *possibly* unearth an odd number of errors (in each of those blocks) which were hidden during the previous pass(es). Bob can hence, return to operate on those blocks and can *possibly* find and correct exactly one more error in each of those blocks using BINARY. Naturally, a new error found in a block formed during the pass $i-1$ (where $i \geq 1$) *possibly* unearths yet another set of odd number of errors in blocks formed during passes 1 to i which contain that error. Consequently, this process of moving back and forth to find and correct errors, which essentially unearths a chain of new errors, sets up a *cascading effect*. This unique feature resulted in this protocol being named as Cascade.

Thus, it is evident that Bob needs to maintain a set O of such blocks and constantly update this set, i.e., add or remove blocks from this set while ensuring that all the blocks in O are unique (i.e., duplicate instances of a block must not be added to O), for each and every pass i (where $i > 1$). Clearly, it's computationally efficient for Bob to operate on the blocks in O in the increasing order of their lengths (i.e,. the smallest block is chosen every time to operate on). We emphasized on the word 'possibly' in the previous paragraph since, the usage of BINARY on the blocks in O comes with a caveat (This had been ignored by the authors who had originally proposed the Cascade protocol [10]). Note that it's very well possible for an *even* number of errors $\epsilon_1, \epsilon_2, \ldots, \epsilon_n$ found in blocks $K_a^i, K_b^i \ldots, K_n^i$ (for some *even* value of $n \leq N$) in pass i (where $i > 1$) to correspond to the same block K_m^j formed during pass j. In such a case, if such a block turns out to be the smallest block, then it must not be operated on by Bob since the block effectively has an even error parity considering that an even number of errors present in it have been corrected in the i^{th} pass. Consequently, Bob must necessarily recheck the error parity of the blocks in O before using BINARY on them. For Bob, a naive way of doing this would be to ask Alice to resend the parity of the block and compare it with the recomputed (by him) parity of the block. However, this is clearly not desirable since this would significantly contribute to an increase in the time overhead of the protocol resulting from the already required numerous parity exchanges taking place between Alice and Bob during the protocol.

A recent study [22] suggested the usage of flag bits as a means to deal with the aforementioned issue. According to their method, flag bits are added to all the blocks formed during the Cascade protocol with their values being set to zero initially. A flag bit equal to 0 indicates the presence of an even number of errors while that equal to 1 indicates the presence of an odd number of errors. Before backtracking to correct errors in previously formed blocks in pass i (where $i > 1$), they check for the presence of an error m in blocks in O and flip the flag bit of a block if an error is found. In this manner, only blocks with flag bits $= 1$ are finally considered for error correction. However, this method clearly imposes an additional storage overhead corresponding to additional flag bits since a lot of blocks are generated during Cascade. Moreover, addition of flag bits is unnecessary for blocks which already have an even error parity; the parities of only those blocks which are present in O need to be computed. Accordingly, here, we present a more computationally efficient solution by making use of only a single variable ('parity_check') which serves as a parity checker only for those blocks which are present in O. 'parity_check' is initialised to 0 and is incremented by 1 for such a given block whenever a previously corrected error (corrected up until then in the i^{th} pass (where $i > 1$) only) is found in it. Clearly, an odd value of 'parity_check' indicates an odd error parity while an even value of the same is indicative of an even error parity. Consequently, only blocks with an odd value of 'parity_check' are operated upon by Bob using BINARY.

The aforementioned process continues until O is empty, i.e., $O = \phi$ for each pass i (where $i > 1$) so as to finally render all the blocks from passes 1 to Ω (the total number of passes of Cascade) with an even error parity. Hence, a number of passes (Ω) of Cascade are able to correct a majority of errors present in Bob's *sifted* key string Q and ultimately render it with a small, even number of errors (possibly even zero).

An algorithmic implementation of Cascade is given in Algorithm 3 (unless otherwise specified, all the computations mentioned in the given algorithm are performed by Bob). Note that in the given algorithm, at a given point of time, Bob only stores the blocks which have an even error parity (with the pass number in which they were originally formed as their key) in the dictionary 'net_block_index_lists'. Blocks which have an odd error parity at a given point of time are operated upon by Bob using BINARY and only the even error parity halves which are successively generated during BINARY are retained. Consequently, later on, only these smaller (as compared to the original big block) blocks need to be dealt with by Bob. Clearly, this would not only lead to higher computational efficiency but would also lead to a significantly lesser number of interactions between Alice and Bob which would consequently speed up the protocol by decreasing its time overhead. A previous study [15] had also suggested a similar optimization (the same optimization was also used in [17] under the name 'Subblock Reuse'), however, their proposition was to constantly store the big blocks and the 2 blocks generated at every step of BINARY and then delete the block being divided further in its next step and add the 2 blocks being generated from it instead.

We do away with all these redundant deletions and subsequent additions in our proposed optimization by directly storing only the even error parity blocks generated during BINARY in 'net_block_index_lists' and in turn, offering a significant speed up over their proposition.

Further, in Algorithm 3, only Bob partitions his sifted key string into blocks at the start of each pass and shuffles its bits at the start of each pass i (where $i > 1$) and then sends (to Alice) the indices of the bits of the key string over which Alice must compute the parities. This optimization reduces Alice's (the server) computational burden as compared to that imposed on Bob (a client). In a Client-Server QKD network, wherein the server operates as a centralized system serving multiple clients simultaneously while each client only communicates directly with the server, such an optimization can approximately balance out the total computational load between the server and the clients leading to a higher computational efficiency of the QKD network [26].

Apart from all the aforementioned optimizations, a significant speedup in computation can further be gained by Alice and Bob if they convert their sifted key strings P and Q to their corresponding integer (i.e., decimal [Base 10]) equivalents: P_{int} and Q_{int}, at the start of the Cascade protocol and use bitwise operators (Right-shift: $\gg$, XOR: $\wedge$ and AND: &) whenever any operations are to be performed on their respective blocks. After Cascade ends, Bob can (if required) simply convert his integer key to the desired binary bit string R (Alice already has P which is unaffected by Cascade, so, no such conversion is required to be performed by her). This optimization also serves to reduce the storage overhead involved (storing 2 integers evidently eats up a lesser amount of memory than storing 2 lists containing 0s and 1s.) and has been implemented in Algorithms 1, 2 and 3 given here (Note that most programming languages don't directly recognize or support performing operations on numbers starting with 0 (as is often the case with binary numbers). Consequently, long lists of 0s and 1s must be used so as to allow any operations to be performed on the corresponding binary numbers. Alternatively, external libraries which allow for the storage and manipulations of binary numbers can be used. However, doing so, evidently decreases the overall efficiency by increasing the memory and computation overhead).

The entire aforementioned discussion also brings to light the fact that Cascade is a highly interactive protocol owing to the numerous exchanges of parities between Alice and Bob over a shared public channel. The high time overhead of this protocol can significantly limit the net key generation rate if a public channel with high latency is used by Alice and Bob. Consequently, it's practical to parallelize Cascade and thereby, process blocks and parities in parallel using pipelining and multi-threading (some amount of parallel processing on similar lines has been implemented in [27]) so as to significantly reduce the time overhead involved.

III. VALUES AND EXPRESSIONS OF KEY PARAMETERS AND QUANTITIES OF CASCADE: A PROBABILISTIC APPROACH

In the previous section, in Algorithm 3, we indicated that the values of k_i ($\forall\ i$) are determined in a manner that depends upon the value of p, i.e., $k_i = f(p)$ ($\forall\ i$). As has already been discussed earlier, parameter optimizations in previous studies [15], [17], [18], [22], [23] have only implied the requirement of a significantly high increase in the number of passes to get better results than Cascade. Moreover, the suggested parameter optimizations have only been backed by some numerical simulations and not by theory and have also been based on certain assumptions observed via a limited number of simulations [14], [17] which may not always hold true in practice. This renders them incapable for generalisation. Moreover, as observed earlier (see Section I), these optimizations, have in part, been motivated by a lack of understanding of the values of the parameters proposed in the original paper on the Cascade protocol [10]. Therefore, we go ahead with the parameter values proposed originally in [10] and now, show/prove in detail how they, in general, i.e., without any such restrictive assumptions, lead to an exponentially decreasing error probability (in Bob's key, when compared to that of Alice's) with the passes of Cascade. We also derive an upper bound on the information leaked (in terms of parity bits) to an eavesdropper, with the passes of Cascade. The only practical assumption we make, is that the QKD protocol involves the usage of a binary symmetric channel ($BSC(p)$) which transmits a string of bits while independently exposing each of the bits to noise with a constant probability p [= QBER].

In the probabilistic analysis that follows, we use standard mathematical notation and a number of new symbols whose meanings are explained as and when they appear in the analysis. For better clarity, however, we provide a consolidated list of all these symbols and their corresponding meanings in Table 1.

We begin with determining $\delta_i(j)$, the probability that $2j$ errors remain in the l^{th} block formed during the 1^{st} pass (K_l^1) after pass $i \geq 1$.

Let's examine the case of $i = 1$. If Y_1 is a random variable denoting the number of bit errors in K_l^1 before the start of pass $i = 1$, then, it's clear that Y_1 follows a Binomial Distribution with k_1 trials and p as the probability of success for each and every trial, i.e., $Y_1 \sim B(k_1, p)$.

Now, there can be two cases when $2j$ errors will remain in K_l^1 after the 1^{st} pass:

- K_l^1 has $2j$ errors before the start of pass $i = 1$: In this case, since K_l^1 will have an *even* error parity, it will be left untouched and no interactive binary search will be performed on it. Consequently, all these $2j$ errors will remain in K_l^1 after pass $i = 1$.
- K_l^1 has $2j+1$ errors before the start of pass $i = 1$: In this case, since K_l^1 will have an *odd* error parity, Alice and Bob will perform an interactive binary search and Bob

Algorithm 3 The Cascade Protocol

Input: P, Q, p, Ω, `random_shuffle()`

global N, P_{int}, Q_{int}, net_block_index_lists
N $\leftarrow |Q|$
P_{int} , Q_{int} $\leftarrow$ `int` $(P, 2)$, `int` $(Q, 2)$
key_index_list $\longleftarrow [0 \ldots \mathrm{N}-1]$
net_block_index_lists $\leftarrow \{\ \}$
for $i \leftarrow 1$ **to** Ω **do**
  if $i > 1$ **then**
    key_index_list $\longleftarrow$ `random_shuffle` (key_index_list)
  end
  k $\leftarrow$ `f` (p)
  block_index_lists $\leftarrow$ [ ]
  net_block_index_lists$[i]$ $\leftarrow$ [ ]
  for $l \leftarrow 1$ **to** $\left\lceil \frac{\mathrm{N}}{\mathrm{k}} \right\rceil$ **do**
    Append key_index_list$[(l-1)k \ldots \min(lk, \mathrm{N}) - 1]$ to block_index_lists
  end
  Bob sends block_index_lists *to Alice*
  Alice computes Alice_parity_list $\leftarrow$ `parity` $(P_{int}$, block_index_lists$)$
  Alice sends Alice_parity_list *to Bob*
  Bob_parity_list $\leftarrow$ `parity` $(Q_{int}$, block_index_lists$)$
  odd_error_parity_block_index_list $\leftarrow$ arg(Alice_parity_list $\neq$ Bob_parity_list)
  for $j \in \left(\left[0 \ldots \left\lceil \frac{\mathrm{N}}{\mathrm{k}} \right\rceil - 1\right] \setminus$ odd_error_parity_block_index_list$\right)$ **do**
    Append block_index_lists$[j]$ to net_block_index_lists$[i]$
  end
  net_index_list_for_bit_flip , complete_index_list_for_bit_flip $\leftarrow$ [ ], [ ]
  for $j \in$ odd_error_parity_block_index_list **do**
    bit_flip_index $\leftarrow$ `binary` (block_index_lists$[j]$, i)
    Q_{int} $\leftarrow Q_{int} \wedge$ `int` $(\text{`0'} \times \text{bit\_flip\_index} + \text{`1'} + \text{`0'} \times (N - \text{bit\_flip\_index} - 1), 2)$
    Append bit_flip_index to net_index_list_for_bit_flip and complete_index_list_for_bit_flip
  end
  if $i > 1$ **then**
    net_check_block_index_lists $\leftarrow$ [ ]
    for bit_flip_index $\in$ net_index_list_for_bit_flip **do**
      for $m \leftarrow i - 1$ **to** 1 **do**
        if bit_flip_index $\in$ sub_list *for some* sub_list $\in$ net_block_index_lists$[m]$ **then**
          if sub_list $\notin$ net_check_block_index_lists **then**
            Append sub_list to net_check_block_index_lists
          end
        end
      end
    end
    while net_check_block_index_lists $\neq \emptyset$ **do**
      check_block_index_list $\leftarrow \min_{\text{length}}$(net_check_block_index_lists)
      check_block_pass_num $\leftarrow$ Key which $\widehat{=}$ check_block_index_list in net_block_index_lists
      Remove check_block_index_list from net_check_block_index_lists
      parity_check $\leftarrow 0$
      if error_index $\in$ check_block_index_list *for some* error_index $\in$ complete_index_list_for_bit_flip **then**
        parity_check $\leftarrow$ parity_check $+ 1$
      end
      if parity_check mod $2 \neq 0$ **then**
        Remove check_block_index_list from net_block_index_lists[check_block_pass_num]
        new_bit_flip_index $\leftarrow$`binary` (check_block_index_list, check_block_pass_num)
        Q_{int} $\leftarrow Q_{int} \wedge$ `int` $(\text{`0'} \times \text{new\_bit\_flip\_index} + \text{`1'} + \text{`0'} \times (N - \text{new\_bit\_flip\_index} - 1), 2)$
        Append new_bit_flip_index to complete_index_list_for_bit_flip
        check_block_new_additions $\leftarrow$ [ ]
        for $m \leftarrow i$ **to** 1 **do**
          if $m \neq$ check_block_pass_num **then**
            if new_bit_flip_index $\in$ sub_list *for some* sub_list $\in$ net_block_index_lists$[m]$ **then**
              Append sub_list to check_block_new_additions
            end
          end
        end
        if check_block_new_additions $\neq \emptyset$ **then**
          for new_add $\in$ check_block_new_additions **do**
            if new_add $\notin$ net_check_block_index_lists **then**
              Append new_add to net_check_block_index_lists
            end
          end
        end
      end
    end
  end
end

TABLE 1. Frequently used notation.

Symbol	Meaning
p	Error Probability (Quantum Bit Error Rate [QBER])
N	Length of Alice's and Bob's Sifted Key Strings
k_i	Block Size in the i^{th} Pass of the Cascade Protocol
K_l^i	l^{th} Block in the i^{th} Pass of the Cascade Protocol
f_i	Random Permutation Function applied to the Indices of the Bits in the Sifted Key in the i^{th} $(i > 1)$ Pass of the Cascade Protocol
ϵ_i	i^{th} Error
Ω	Total Number of Passes of the Cascade Protocol
$\delta_i(j)$	Probability that $2j$ Errors remain in K_l^1 after Pass $i \geqslant 1$ of the Cascade Protocol
Y_1	Random Variable denoting the Number of Bit Errors in K_l^1 before the Start of Pass $i = 1$ of the Cascade Protocol
E_i	Expected Number of Errors in K_l^1 at the End of the i^{th} Pass of the Cascade Protocol
γ_i	Probability of correcting at least 2 Errors at Pass $i > 1$ in a Block K_l^1 which still contains Errors after the Completion of Pass $i - 1$ of the Cascade Protocol
X_i	Random Variable denoting the Number of Errors remaining in K_l^1 after Pass $i > 1$ of the Cascade Protocol
$I(\omega)$	Amount of Information leaked after ω Passes per Block of Length k_1, i.e., per Block K_l^1 by the Cascade Protocol to a potential Eavesdropper in terms of the Number of Parity Bits
$I_{net}(\omega)$	Total Amount of Information leaked after ω Passes by the Cascade Protocol to a potential Eavesdropper in terms of the Number of Parity Bits

will be able to correct exactly 1 bit error in K_l^1 at the end of the 1st pass. Consequently, $2j$ errors will again remain in K_l^1 after pass $i = 1$.

Thus,

$$\delta_1(j) = P(Y_1 = 2j) + P(Y_1 = 2j + 1). \tag{1}$$

Now, let E_i be the expected number of errors in K_l^1 at the end of the i^{th} pass. Clearly, for $i = 1$, we have that:

$$E_1 = \sum_{j=0}^{\lfloor \frac{k_1}{2} \rfloor} 2j\delta_1(j).$$

Note that the upper limit of the summation is $\lfloor \frac{k_1}{2} \rfloor$ since $2j$ can at maximum be equal to k_1 in a block K_l^1 of k_1 bits.

Since the contribution from $j = 0$ to E_1 is 0,

$$E_1 = \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} 2j\delta_1(j). \tag{2}$$

Using (1),

$$E_1 = \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} 2j(P(Y_1 = 2j) + P(Y_1 = 2j + 1)).$$

Since $Y_1 \sim B(k_1, p)$,

$$P(Y_1 = j) = {}^{k_1}C_j p^j (1-p)^{k_1 - j}.$$

Consequently, we have that:

$$\begin{aligned}
E_1 &= \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} 2j^{k_1}C_{2j}p^{2j}(1-p)^{k_1-2j} \\
&\quad + \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} 2j^{k_1}C_{2j+1}p^{2j+1}(1-p)^{k_1-2j-1}. \\
\implies E_1 &= \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} 2j^{k_1}C_{2j}p^{2j}(1-p)^{k_1-2j} \\
&\quad + \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} (2j+1)^{k_1}C_{2j+1}p^{2j+1}(1-p)^{k_1-2j-1} \\
&\quad + {}^{k_1}C_1 p^1 (1-p)^{k_1-1} \\
&\quad - \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} {}^{k_1}C_{2j+1}p^{2j+1}(1-p)^{k_1-2j-1} \\
&\quad - {}^{k_1}C_1 p^1 (1-p)^{k_1-1}. \\
\implies E_1 &= \sum_{\substack{2j=0 \\ 2j+=2}}^{2\lfloor \frac{k_1}{2} \rfloor} 2j^{k_1}C_{2j}p^{2j}(1-p)^{k_1-2j} \\
&\quad + \sum_{\substack{2j=0 \\ 2j+=2}}^{2\lfloor \frac{k_1}{2} \rfloor} (2j+1)^{k_1}C_{2j+1}p^{2j+1}(1-p)^{k_1-2j-1} \\
&\quad - \sum_{\substack{2j=0 \\ 2j+=2}}^{2\lfloor \frac{k_1}{2} \rfloor} {}^{k_1}C_{2j+1}p^{2j+1}(1-p)^{k_1-2j-1}.
\end{aligned} \tag{3}$$

Clearly, the 1st 2 terms in (3) cover all the values of Y_1 from 0 to k_1. Consequently, we have that:

$$\begin{aligned}
&\sum_{\substack{2j=0 \\ 2j+=2}}^{2\lfloor \frac{k_1}{2} \rfloor} 2j^{k_1}C_{2j}p^{2j}(1-p)^{k_1-2j} \\
&\quad + \sum_{\substack{2j=0 \\ 2j+=2}}^{2\lfloor \frac{k_1}{2} \rfloor} (2j+1)^{k_1}C_{2j+1}p^{2j+1}(1-p)^{k_1-2j-1} \\
&= \sum_{j=0}^{k_1} jP(Y_1 = j) \\
&= E(Y_1) \\
&= k_1 p.
\end{aligned} \tag{4}$$

Further, the 3rd term in (3) is clearly the sum of odd terms in the binomial expansion of $((1-p)+p)^{k_1}$.

Consequently, since:

$$((1-p)+p)^{k_1} = \sum_{j=0}^{k_1} {}^{k_1}C_j (1-p)^{k_1-j} p^j$$

and

$$((1-p)-p)^{k_1} = \sum_{j=0}^{k_1} {}^{k_1}C_j (1-p)^{k_1-j} (-1)^j p^j,$$

we have that:

$$\begin{aligned}
&((1-p)+p)^{k_1} - ((1-p)-p)^{k_1} \\
&= \sum_{j=0}^{k_1} {}^{k_1}C_j [1-(-1)^j] p^j (1-p)^{k_1-j} \\
&= 2 \sum_{j \text{ odd}} {}^{k_1}C_j p^j (1-p)^{k_1-j}.
\end{aligned}$$

Consequently,

$$\begin{aligned}
\sum_{j \text{ odd}} {}^{k_1}C_j p^j (1-p)^{k_1-j} &= \frac{((1-p)+p)^{k_1} - ((1-p)-p)^{k_1}}{2} \\
&= \frac{(1)^{k_1} - (1-2p)^{k_1}}{2} \\
&= \frac{1-(1-2p)^{k_1}}{2}.
\end{aligned}$$

Thus,

$$\begin{aligned}
\sum_{\substack{2j=0 \\ 2j+=2}}^{2\lfloor \frac{k_1}{2} \rfloor} {}^{k_1}C_{2j+1} p^{2j+1} (1-p)^{k_1-2j-1} &\equiv \sum_{j \text{ odd}} {}^{k_1}C_j p^j (1-p)^{k_1-j} \\
&= \frac{1-(1-2p)^{k_1}}{2}. \qquad (5)
\end{aligned}$$

Putting (4) and (5) into (3) yields:

$$E_1 = k_1 p - \frac{1-(1-2p)^{k_1}}{2}. \qquad (6)$$

Now, considering that the permutation functions f_i for $i > 1$ are chosen randomly from the set $\{f \mid f : [0, \ldots, (N-1)] \to [1, \ldots, \lceil \frac{N}{k_i} \rceil]\}$, for $N \to \infty$, it's possible to determine a lower bound on the probability γ_i of correcting at least 2 errors at pass $i > 1$ in a block K_l^1 which still contains errors after the completion of the pass $i-1$.

We have that:

$$\gamma_i = P\left(\frac{\text{correcting } \geq 2 \text{ errors in } K_l^1 \text{ at pass } i > 1}{K_l^1 \text{ contains errors at the end of pass } i-1} \right).$$

Given that K_l^1 does contain some errors at the end of pass $i-1$, the minimum possible number of errors in K_l^1 can only be 2 since only an even number of errors can possibly remain in a block after a pass. Consequently,

$$\gamma_i = 1 - P\left(\frac{\text{not correcting any errors in } K_l^1 \text{ at pass } i > 1}{K_l^1 \text{ contains errors at the end of pass } i-1} \right).$$

An upper bound for $P\left(\frac{\text{not correcting any errors in } K_l^1 \text{ at pass } i > 1}{K_l^1 \text{ contains errors at the end of pass } i-1} \right)$ can be found out by considering the following 2 cases corresponding to probabilities which can possibly be equal to it:

- Case 1: $P\left(\frac{E_1}{F_1}\right)$
  where,
  $E_1 = \epsilon_1$ and ϵ_2 are both present in K_l^i and $\therefore$, K_l^i possesses an even error parity at pass $i > 1$
  and,
  $F_1 = \exists$ (among other possible errors) errors ϵ_1 and ϵ_2 in K_l^1 at the end of pass $i-1$.
- Case 2: $P\left(\frac{E_2}{F_2}\right)$
  where,
  $E_1 = \epsilon_1$ is present in K_l^i and ϵ_2 is present in K_m^i and both K_l^i and K_m^i possess an even error parity at pass $i > 1$
  and,
  $F_1 = \exists$ (among other possible errors) errors ϵ_1 and ϵ_2 in K_l^1 at the end of pass $i-1$.

Clearly, the Case 2 probability is greater as compared to the Case 1 probability, since ϵ_1 and ϵ_2 have a larger probability of belonging to different blocks in pass $i > 1$ rather than the same block due to shuffling of the indices of the bits of the sifted key at the start of pass $i > 1$. Consequently, it's possible to say that:

$$P\left(\frac{\text{not correcting any errors in } K_l^1 \text{ at pass } i > 1}{K_l^1 \text{ contains errors at the end of pass } i-1} \right) \leq P\left(\frac{E_2}{F_2}\right).$$

Hence, we have that:

$$\gamma_i \geq 1 - P\left(\frac{E_2}{F_2}\right).$$

Now, we can consider a possible case (Case A) of equivalence for $P\left(\frac{E_2}{F_2}\right)$:

$$\begin{aligned}
&P(\text{Case 2}) \\
&= P\left(\frac{\exists \text{ exactly 1 more error apart from } \epsilon_1 \text{ in } K_l^i}{\exists\ \epsilon_1 \text{ in } K_l^i} \right) \\
&\quad \times P\left(\frac{\exists \text{ exactly 1 more error apart from } \epsilon_2 \text{ in } K_m^i}{\exists\ \epsilon_2 \text{ in } K_m^i} \right).
\end{aligned}$$

Since blocks are treated in the same manner, the 2 conditional probabilities on the R.H.S. of the aforementioned expression would be equal. Therefore,

$$\begin{aligned}
&P\left(\frac{E_2}{F_2}\right) \\
&= \left[P\left(\frac{\exists \text{ exactly 1 more error apart from } \epsilon_1 \text{ in } K_l^i}{\exists\ \epsilon_1 \text{ in } K_l^i} \right) \right]^2.
\end{aligned}$$

Apart from case A, there is another possibility (Case B) of equivalence for $P\left(\frac{E_2}{F_2}\right)$:

$$P\left(\frac{E_2}{F_2}\right) = \left[P\left(\frac{\exists \text{ at least 1 more error apart from } \epsilon_1 \text{ in } K_l^i}{\exists\ \epsilon_1 \text{ in } K_l^i}\right)\right]^2.$$

Clearly,

$$P\left(\frac{\exists \text{ at least 1 more error apart from } \epsilon_1 \text{ in } K_l^i}{\exists\ \epsilon_1 \text{ in } K_l^i}\right) \geq P\left(\frac{\exists \text{ exactly 1 more error apart from } \epsilon_1 \text{ in } K_l^i}{\exists\ \epsilon_1 \text{ in } K_l^i}\right).$$

Hence, $P\left(\frac{E_2}{F_2}\right)$ can be upper bounded using P (Case B) and we get that:

$$\begin{aligned}\gamma_i &\geq 1 - \left[P\left(\frac{\exists \text{ at least 1 more error apart from } \epsilon_1 \text{ in } K_l^i}{\exists\ \epsilon_1 \text{ in } K_l^i}\right)\right]^2 \\ &= 1 - \left[1 - P\left(\frac{\exists \text{ zero more errors apart from } \epsilon_1 \text{ in } K_l^i}{\exists\ \epsilon_1 \text{ in } K_l^i}\right)\right]^2 \\ &= 1 - \left[1 - P\left(\exists \text{ exactly 1 error in } K_l^i\right)\right]^2.\end{aligned}$$

Now, E_{i-1} is the expected number of errors in K_l^1 after pass $i-1$ (where $i > 1$) and $\lceil \frac{N}{k_1} \rceil$ is the total number of blocks in the 1st pass. Hence, $\lceil \frac{N}{k_1} \rceil E_{i-1}$ is the expected total number of errors in the entire key after pass $i-1$.

Consequently,

$$P\left(\exists \text{ exactly 1 error in } K_l^i\right) = P\left(\exists \left(\left\lceil \frac{N}{k_1} \right\rceil E_{i-1} - 1\right) \text{ errors outside } K_l^i\right).$$

Now, k_i bits of the sifted key lie in the block K_l^i. Hence, $N - k_i$ bits of the sifted key lie outside the block K_l^i.

Consequently,

$$P\left(\exists \text{ 1 error outside } K_l^i\right) = \frac{N - k_i}{N}.$$

Since (by assumption) the probabilities of the occurrence of errors in a block are independent of one another and that these probabilities have a constant value for all the errors, we get that:

$$P\left(\exists \left(\left\lceil \frac{N}{k_1} \right\rceil E_{i-1} - 1\right) \text{ errors outside } K_l^i\right) = \left(\frac{N - k_i}{N}\right)^{\lceil \frac{N}{k_1} \rceil E_{i-1} - 1}.$$

Since we are considering the case when $N \to \infty$,

$$\left\lceil \frac{N}{k_1} \right\rceil E_{i-1} - 1 \approx \frac{N}{k_1} E_{i-1} - 1 \approx \frac{N}{k_1} E_{i-1}.$$

Hence,

$$\begin{aligned}\gamma_i &\geq 1 - \left[1 - \left(\frac{N - k_i}{N}\right)^{\frac{N}{k_1} E_{i-1}}\right]^2 \\ &= 1 - \left[1 - \left(1 - \frac{k_i}{N}\right)^{\frac{N}{k_1} E_{i-1}}\right]^2. \qquad (7)\end{aligned}$$

In order to simplify the R.H.S. of the above expression further since $N \to \infty$ here, let's consider the evaluation of the following limit:

$$\lim_{N \to \infty} \left(1 - \frac{k_i}{N}\right)^{\frac{N}{k_1} E_{i-1}}.$$

To do this, let's start by considering the following form of the above limit:

$$\lim_{y \to \infty} \left(1 - \frac{x}{y}\right)^{ay}.$$

Using the Binomial Theorem, we know that:

$$\left(1 - \frac{x}{y}\right)^{ay} = \sum_{k=0}^{ay} {}^{ay}C_k (1)^{ay-k} (-1)^k \left(\frac{x}{y}\right)^k.$$

Consequently,

$$\left(1 - \frac{x}{y}\right)^{ay} = 1 - ay\left(\frac{x}{y}\right) + \left(\frac{ay(ay-1)}{2}\right)\left(\frac{x}{y}\right)^2 + \ldots + (-1)^{ay}\left(\frac{x}{y}\right)^{ay}.$$

In the limiting case when $y \to \infty$, we thus, get that:

$$\lim_{y \to \infty} \left(1 - \frac{x}{y}\right)^{ay} = 1 - ax + \frac{a^2x^2}{2!} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n (ax)^n}{n!} = e^{-ax}.$$

Hence,

$$\lim_{N \to \infty} \left(1 - \frac{k_i}{N}\right)^{\frac{N}{k_1} E_{i-1}} = e^{-\frac{k_i E_{i-1}}{k_1}}. \qquad (8)$$

Consequently, for $N \to \infty$, putting (8) in (7) yields:

$$\gamma_i \gtrapprox 1 - \left[1 - e^{-\frac{k_i E_{i-1}}{k_1}}\right]^2. \qquad (9)$$

We now wish to bound $\delta_i(j)$ using γ_i for $i > 1$.

In order to do this, let's consider a random variable X_i which denotes the number of errors remaining in K_l^1 after pass $i > 1$. Consequently,

$$\delta_i(j) = P((X_{i-1} = 2j) \cap (X_i = 2j)) + P((X_{i-1} - X_i > 0) \cap (X_i = 2j)).$$

This is because $2j$ errors can remain in K_l^1 after pass $i > 1$ when either K_l^1 had $2j$ or $> 2j$ number of errors in the previous pass.

Now,

$$\begin{aligned} &P((X_{i-1} - X_i > 0) \cap (X_i = 2j)) \\ &= P(X_i = 2j) \times P\left(\frac{X_{i-1} > X_i}{X_i = 2j}\right) \\ &= P(X_i = 2j) \times P(X_{i-1} > 2j). \end{aligned}$$

Since $P(X_i = 2j) \in [0, 1]$,

$$P(X_i = 2j) \times P(X_{i-1} > 2j) \leq P(X_{i-1} > 2j).$$

Consequently,

$$\begin{aligned} \delta_i(j) &\leq P((X_{i-1} = 2j) \cap (X_i = 2j)) + P(X_{i-1} > 2j) \\ &= P((X_i = 2j) \cap (X_{i-1} = 2j)) + P(X_{i-1} > 2j) \\ &= P(X_{i-1} = 2j) \times P\left(\frac{X_i = 2j}{X_{i-1} = 2j}\right) + P(X_{i-1} > 2j). \end{aligned}$$

By definition, we have that:

$$P(X_{i-1} = 2j) = \delta_{i-1}(j),$$
$$P\left(\frac{X_i = 2j}{X_{i-1} = 2j}\right) = 1 - \gamma_i,$$

and

$$P(X_{i-1} > 2j) = \sum_{l=j+1}^{\lfloor \frac{k_1}{2} \rfloor} \delta_{i-1}(j).$$

Thus,

$$\delta_i(j) \leq \sum_{l=j+1}^{\lfloor \frac{k_1}{2} \rfloor} \delta_{i-1}(l) + \delta_{i-1}(j)(1 - \gamma_i). \tag{10}$$

Now, consider that k_1 is chosen such that:

$$\sum_{l=j+1}^{\lfloor \frac{k_1}{2} \rfloor} \delta_1(l) \leq \frac{1}{4}\delta_1(j), \tag{11}$$

and let $k_i = 2k_{i-1}$ for $i > 1$. Accordingly, we have that:

$$k_i = 2^{i-1}k_1. \tag{12}$$

Now, putting (9) in (10) gives:

$$\delta_i(j) \leq \sum_{l=j+1}^{\lfloor \frac{k_1}{2} \rfloor} \delta_{i-1}(l) + \delta_{i-1}(j)\left(1 - e^{-\frac{k_i E_{i-1}}{k_1}}\right)^2.$$

Using (12), the above expression simplifies to:

$$\delta_i(j) \leq \sum_{l=j+1}^{\lfloor \frac{k_1}{2} \rfloor} \delta_{i-1}(l) + \delta_{i-1}(j)\left(1 - e^{-2^{i-1}E_{i-1}}\right)^2. \tag{13}$$

In addition to satisfying (11), let's consider that the choice of k_1 is such that it satisfies the following inequality:

$$E_1 \leq -\frac{\ln(\frac{1}{2})}{2}. \tag{14}$$

Let's now analyse the case when $i = 2$ and start with (9). Thus, using (12), we get that:

$$\gamma_2 \gtrapprox 1 - \left[1 - e^{-2E_1}\right]^2.$$

Using (14),

$$\gamma_2 \gtrapprox 1 - \left[1 - e^{\ln(\frac{1}{2})}\right]^2 = \frac{3}{4}. \tag{15}$$

When $i = 2$, (10) becomes:

$$\delta_2(j) \leq \sum_{l=j+1}^{\lfloor \frac{k_1}{2} \rfloor} \delta_1(l) + \delta_1(j)(1 - \gamma_2).$$

Using (11),

$$\delta_2(j) \leq \frac{1}{4}\delta_1(j) + \delta_1(j)(1 - \gamma_2) = \left(\frac{5}{4} - \gamma_2\right)\delta_1(j).$$

Using (15),

$$\begin{aligned} &\delta_2(j) \leq \left(\frac{5}{4} - \frac{3}{4}\right)\delta_1(j) \\ &\implies \delta_2(j) \leq \frac{\delta_1(j)}{2}. \end{aligned} \tag{16}$$

By definition,

$$E_2 = \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} 2j\delta_2(j).$$

Using (16),

$$E_2 \leq \left(\frac{1}{2}\right) \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} 2j\delta_1(j).$$

Using (2),

$$E_2 \leq \frac{E_1}{2}. \tag{17}$$

Similarly, we can also analyse the case when $i = 3$ by starting with (9) and using (12). We get that:

$$\gamma_3 \gtrapprox 1 - \left[1 - e^{-4E_2}\right]^2.$$

Using (17),

$$\gamma_3 \gtrapprox 1 - \left[1 - e^{-\frac{4E_1}{2}}\right]^2 = 1 - \left[1 - e^{-2E_1}\right]^2.$$

Using (14),

$$\gamma_3 \gtrapprox 1 - \left[1 - e^{\ln(\frac{1}{2})}\right]^2 = \frac{3}{4}. \tag{18}$$

When $i = 3$, (10) becomes:

$$\delta_3(j) \leq \sum_{l=j+1}^{\lfloor \frac{k_1}{2} \rfloor} \delta_2(l) + \delta_2(j)(1 - \gamma_3).$$

Using (16),

$$\delta_3(j) \leq \frac{1}{2}\left(\sum_{l=j+1}^{\lfloor \frac{k_1}{2} \rfloor} \delta_1(l) + \delta_1(j)(1-\gamma_3)\right).$$

Using (11),

$$\delta_3(j) \leq \frac{1}{2}\left(\frac{1}{4}\delta_1(j) + \delta_1(j)(1-\gamma_3)\right) = \frac{1}{2}\left(\frac{5}{4} - \gamma_3\right)\delta_1(j)$$

Using (18),

$$\delta_3(j) \leq \frac{1}{2}\left(\frac{5}{4} - \frac{3}{4}\right)\delta_1(j)$$

$$\implies \delta_3(j) \leq \frac{\delta_1(j)}{4}. \tag{19}$$

By definition,

$$E_3 = \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} 2j\delta_3(j).$$

Using (19),

$$E_3 \leq \left(\frac{1}{4}\right)\sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} 2j\delta_1(j).$$

Using (2),

$$E_3 \leq \frac{E_1}{4}. \tag{20}$$

One can easily extend this analysis $\forall\, i > 3$ as well and find that the following results will hold true $\forall\, i > 1$:

$$\gamma_i \gtrapprox \frac{3}{4}. \tag{21}$$

$$\delta_i(j) \leq \frac{\delta_1(j)}{2^{i-1}}. \tag{22}$$

$$E_i \leq \frac{E_1}{2^{i-1}}. \tag{23}$$

From (22) and (23), it's quite clear that the probability that a block K_l^1 has ≥ 1 errors decreases exponentially with respect to the number of passes when the choice of k_1 is such that it satisfies (11) and (14) and $k_i = 2k_{i-1}$ for $i > 1$.

It's also possible to derive an upper bound on the amount of information $[I(\omega)]$ leaked after ω passes per block of length k_1, i.e., per block K_l^1 (with all the k_i values chosen as discussed above) by the Cascade protocol to a potential eavesdropper in terms of the number of parity bits.

In accordance with the aforementioned notations, it's expected that:

$$I(\omega) \leq 1 + \sum_{j=0}^{\lfloor \frac{k_1}{2} \rfloor} P(Y_1 = 2j+1)\lceil \log_2 k_1 \rceil + \sum_{l=2}^{\omega} E_l \lceil \log_2 k_1 \rceil + \frac{1}{2}\left(\sum_{l=2}^{\omega} E_l\right). \tag{24}$$

The reasons for the inclusion of the 4 terms in the R.H.S of the above expression are as follows:

- 1: Because 1 parity value corresponding to the parity of the entire block K_l^1 is necessarily exchanged between Alice and Bob before the start of the Cascade protocol in order to ascertain whether K_l^1 has an odd error parity or not.
- $\sum_{j=0}^{\lfloor \frac{k_1}{2} \rfloor} P(Y_1 = 2j+1)\lceil \log_2 k_1 \rceil$: It must be noted that whenever (to start with) block K_l^1 has an odd number of errors, Alice and Bob will perform an interactive binary search so as to find and hence, correct exactly 1 error in K_l^1. This interactive binary search entails the exchange of $\lceil \log_2 k_1 \rceil$ bits between Alice and Bob. Consequently, in order to compute the expected number of parity bits that can possibly be leaked, we multiply the probability that the block K_l^1 has an odd number of errors before the start of the pass $i = 1$ $[= P(Y_1 = 2j+1)]$ with the number of parity bits that are leaked when an odd number of errors are present in K_l^1 before the start of the pass $i = 1$ $[= \lceil \log_2 k_1 \rceil]$ and sum the resultant product over all the possible values of the odd number of errors that can be present in K_l^1.
- $\sum_{l=2}^{\omega} E_l \lceil \log_2 k_1 \rceil$: Assuming that all the expected number of errors still present in K_l^1 (after the 1st pass) till the ω^{th} pass are corrected by Alice and Bob in ω passes (this is clearly an upper bound on the number of errors corrected in ω passes (in part; the other part of the bound is covered in the 4th term)), $\lceil \log_2 k_1 \rceil$ parity bits will be exchanged between Alice and Bob corresponding to interactive binary searches performed on K_l^1 for each of the errors. Consequently, $\sum_{l=2}^{\omega} E_l \lceil \log_2 k_1 \rceil$ will be the total number of leaked parity bits in passes $2, \ldots, \omega$.
- $\frac{1}{2}\left(\sum_{l=2}^{\omega} E_l\right)$: In line with the above mentioned assumption, apart from $\lceil \log_2 k_1 \rceil$ parity bits exchanged between Alice and Bob corresponding to interactive binary searches performed on K_l^1 for each of the errors, parity bits corresponding to the recomputed parity of the entire block K_l^1 in passes $2, \ldots, \omega$ will also be exchanged between Alice and Bob in order to ascertain whether/not K_l^1 has an odd error parity. This will clearly be done for half the expected number of errors that are corrected in a given pass $\forall$ passes $2, \ldots, \omega$ since errors are corrected two by two in K_l^1 in the passes wherein $i > 1$.

We now proceed on with the simplification of the expression for $I(\omega)$ presented in (24).

As shown earlier, since $Y_1 \sim B(k_1, p)$,

$$P(Y_1 = 2j+1) = {}^{k_1}C_{2j+1}p^{2j+1}(1-p)^{k_1-2j-1}.$$

Using (5),

$$\begin{aligned}\sum_{j=0}^{\lfloor \frac{k_1}{2} \rfloor} P(Y_1 = 2j+1) &= \sum_{j=0}^{\lfloor \frac{k_1}{2} \rfloor} {}^{k_1}C_{2j+1}p^{2j+1}(1-p)^{k_1-2j-1} \\ &= \sum_{\substack{2j=0 \\ 2j+=2}}^{2\lfloor \frac{k_1}{2} \rfloor} {}^{k_1}C_{2j+1}p^{2j+1}(1-p)^{k_1-2j-1} \\ &= \frac{1-(1-2p)^{k_1}}{2}. \end{aligned} \tag{25}$$

By definition,

$$E_i = \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} 2j\delta_i(j).$$

Using (22),

$$E_i \leq \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} 2j \left(\frac{\delta_1(j)}{2^{i-1}} \right). \tag{26}$$

Putting (25) and (26) in (24) yields:

$$\begin{aligned} I(\omega) \leq 1 &+ \frac{1-(1-2p)^{k_1}}{2} \lceil \log_2 k_1 \rceil \\ &+ 2 \sum_{l=2}^{\omega} \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} \frac{j\delta_1(j)}{2^{l-1}} \lceil \log_2 k_1 \rceil \\ &+ \frac{1}{2} \left(2 \sum_{l=2}^{\omega} \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} \frac{j\delta_1(j)}{2^{l-1}} \right). \end{aligned} \tag{27}$$

Let's now derive an upper bound for the 4$^{\text{th}}$ term in the above inequality:

$$\begin{aligned} \frac{1}{2} \left(2 \sum_{l=2}^{\omega} \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} \frac{j\delta_1(j)}{2^{l-1}} \right) &= \sum_{l=2}^{\omega} \frac{1}{2^{l-1}} \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} j\delta_1(j) \\ &= \sum_{l=2}^{\omega} \frac{1}{2^{l-1}} \left(\frac{E_1}{2} \right). \end{aligned}$$

Using (6),

$$\frac{1}{2} \left(2 \sum_{l=2}^{\omega} \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} \frac{j\delta_1(j)}{2^{l-1}} \right) = \frac{1}{2} \sum_{l=2}^{\omega} \frac{1}{2^{l-1}} \left(k_1 p - \frac{1-(1-2p)^{k_1}}{2} \right).$$

Clearly, $(1-2p)^{k_1} \leq 1$.
Consequently,

$$\begin{aligned} \frac{1}{2} \left(2 \sum_{l=2}^{\omega} \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} \frac{j\delta_1(j)}{2^{l-1}} \right) &\leq \frac{1}{2} \sum_{l=2}^{\omega} \frac{1}{2^{l-1}} \left(k_1 p - \frac{1}{2} + \frac{1}{2} \right) \\ &= \frac{1}{2} \sum_{l=2}^{\omega} \frac{k_1 p}{2^{l-1}}. \end{aligned}$$

Now, it was already specified in the beginning that we choose k_1 in a manner that depends on p. Let's analyse this a bit more closely.

Consider that $p = \frac{e}{100}$. This means that on average, we have e errors per 100 bits of the sifted key. Further, this implies that a block of $\frac{100}{e}$ $(= \frac{1}{p})$ bits will contain 1 error on average. Thus, $k_1 = \frac{1}{p}$ seems to be a good choice of k_1 since all the errors in the key can possibly get corrected faster considering that more number of blocks would have an odd error parity to start with itself. This was the value of k_1 that had been used in [16] and [17]. However, since k_1 must necessarily satisfy constraints (11) and (14), in practice, k_1 turns out to be $< \frac{1}{p}$ (as we show later on). Consequently, in general, we wouldn't require $k_1 > \frac{1}{p}$ at all and hence, we can consider that $k_1 \leq \frac{1}{p}$.

Hence, we have that:

$$\begin{aligned} \frac{1}{2} \left(2 \sum_{l=2}^{\omega} \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} \frac{j\delta_1(j)}{2^{l-1}} \right) &\leq \frac{1}{2} \sum_{l=2}^{\omega} \frac{\left(\frac{1}{p}\right)p}{2^{l-1}} \\ &= \frac{1}{2} \sum_{l=2}^{\omega} \frac{1}{2^{l-1}} \\ &\leq \frac{1}{2} \sum_{l=1}^{\infty} \frac{1}{2^{l-1}} \\ &= \frac{1}{2} \left[1 + \frac{1}{2} + \frac{1}{4} + \ldots \right] \\ &= \frac{1}{2} \left[\frac{1}{\left(1 - \frac{1}{2}\right)} \right] \\ &= 1. \end{aligned} \tag{28}$$

Putting (28) in (27) yields:

$$\begin{aligned} I(\omega) \leq\ & 1 + \frac{1-(1-2p)^{k_1}}{2} \lceil \log_2 k_1 \rceil \\ &+ 2 \sum_{l=2}^{\omega} \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} \frac{j\delta_1(j)}{2^{l-1}} \lceil \log_2 k_1 \rceil + 1. \\ \implies I(\omega) \leq\ & 2 + \frac{1-(1-2p)^{k_1}}{2} \lceil \log_2 k_1 \rceil \\ &+ 2 \sum_{l=2}^{\omega} \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} \frac{j\delta_1(j)}{2^{l-1}} \lceil \log_2 k_1 \rceil. \end{aligned} \tag{29}$$

Since the number of blocks of length $k_1 = \lceil \frac{N}{k_1} \rceil$, the total amount of information $[I_{net}(\omega)]$ leaked after ω passes is:

$$\implies I_{net}(\omega) = \left\lceil \frac{N}{k_1} \right\rceil I(\omega).$$

Consequently,

$$\begin{aligned} \implies I_{net}(\omega) \leq & \left\lceil \frac{N}{k_1} \right\rceil \left(2 + \frac{1-(1-2p)^{k_1}}{2} \lceil \log_2 k_1 \rceil \right) \\ &+ \left\lceil \frac{N}{k_1} \right\rceil \left(2 \sum_{l=2}^{\omega} \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} \frac{j\delta_1(j)}{2^{l-1}} \lceil \log_2 k_1 \rceil \right). \end{aligned} \tag{30}$$

Now, while we do wish to choose the values of k_i such that they lead to an exponentially decreasing error probability $\delta_i(j)$ with respect to the number of passes of Cascade, we also want to minimize, *as much as possible*, the amount of information

1001011001010011011101101101101011011011010011010110100001010101101011101001100101011010011010111011

(a) Bit String

74437845748267780162814356037 9

(b) Corresponding Integer Equivalent

FIGURE 3. Alice's sifted key string.

1011011101000011011001100101101011011011010011010011100001010101101111100001100101011010011010111010

(a) Bit String

907475810953774330838997378746

(b) Corresponding Integer Equivalent

FIGURE 4. Bob's sifted key string.

BLOCK NUMBER	BLOCK	PARITY MISMATCH
1	0 1 2 3 4 5 6 7 8	✗
2	9 10 11 12 13 14 15 16 17	✓
3	18 19 20 21 22 23 24 25 26	✗
4	27 28 29 30 31 32 33 34 35	✗
5	36 37 38 39 40 41 42 43 44	✗
6	45 46 47 48 49 50 51 52 53	✗
7	54 55 56 57 58 59 60 61 62	✗
8	63 64 65 66 67 68 69 70 71	✓
9	72 73 74 75 76 77 78 79 80	✓
10	81 82 83 84 85 86 87 88 89	✗
11	90 91 92 93 94 95 96 97 98	✗
12	99	✓

FIGURE 5. Blocks, their corresponding block numbers and error parities at the start of Pass 1 of the cascade protocol [Undetected error locations are indicated by the colour yellow].

$[I_{net}(\omega)]$ being leaked to a potential eavesdropper, say, Eve. This is because a larger of number of parity bits leaked to Eve makes it more easier for her to construct the key being shared by only Alice and Bob.

From (30), it's evident that, for a fixed value of p, as k_1 increases, $I_{net}(\omega)$ decreases.

Consequently, we choose the largest value of k_1 that satisfies (11) and (14) and set $k_i = 2k_{i-1}$ $(\forall\, i > 1)$. This thus, helps us in achieving a dual objective: an exponentially decreasing error probability with the passes of Cascade (since k_1 satisfies (11) and (14) and $k_i = 2k_{i-1}$ $(\forall\, i > 1)$) and only the minimum possible information being leaked to a potential eavesdropper (largest k_1 implies least $I_{net}(\omega)$).

On the basis of some empirical tests, the original authors of Cascade [10] recommended 4 passes of the protocol in total $(\Omega = 4)$, i.e., $\omega = 4$, for obtaining a negligible amount of error in the final key strings being possessed by Alice and Bob. Table 2 shows the values of k_1 and $I(\omega)$ for $\omega = 4$ and for $p \in \{0.01, 0.03, 0.05, 0.075, 0.10, 0.125, 0.15\}$ which have been obtained by following the procedure outlined in the previous paragraph and by setting $\omega = 4$ in (29) respectively. For $p \in \{0.01, 0.05, 0.10, 0.15\}$, it can be easily seen that the values of k_1 are exactly equal to, and the values of $I(4)$ are approximately equal to their corresponding values in Table 1 of [10]. It also seems reasonable to say that $k_1 \approx 0.73/p$, as was stated in [24].

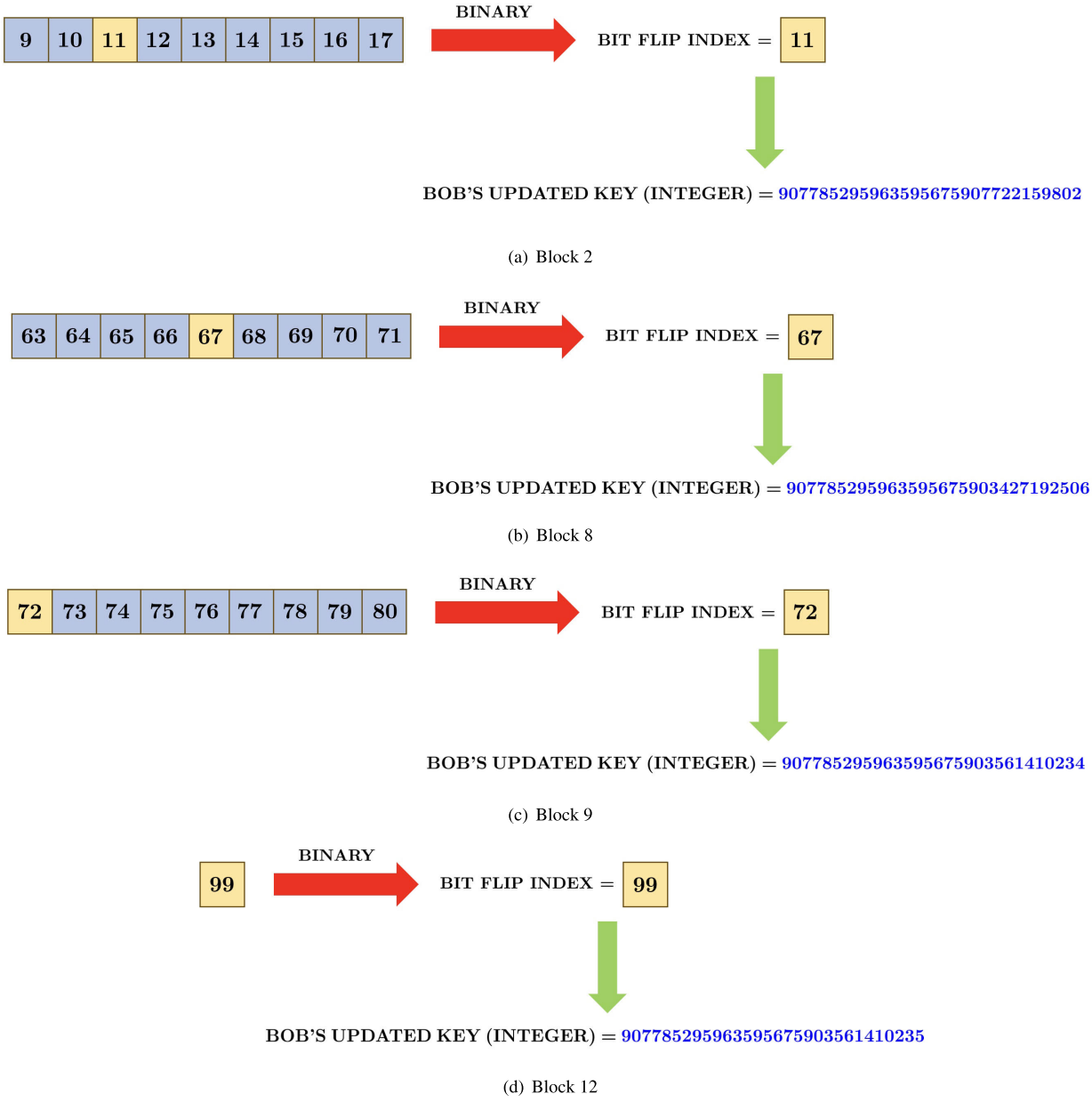


FIGURE 6. Detection and correction of errors using BINARY in Pass 1 of the cascade protocol [Undetected error locations are indicated by the colour yellow].

TABLE 2. Values of important parameters related to the cascade protocol.

p	k_1	$I(4)$
0.01	73	6.809
0.03	24	5.392
0.05	14	4.643
0.075	9	4.555
0.10	7	3.986
0.125	6	4.123
0.15	5	4.125

IV. DEVIATIONS FROM RELATED PROBABILISTIC ANALYSES

While our probabilistic analysis of the Cascade protocol builds up on the one given originally in [10] and follows some expressions and notations given in [10] and [16], it differs from them in that it extensively clarifies the math and is backed by physical interpretation at every step of the analysis. Certain aspects concerning the flow (order) of the analysis, etc. in which it differs from the analyses/proofs given in [10] and [16] are quite important to note and we now begin to enumerate them.

Notwithstanding certain major errors such as stating that $E_i = \sum_{j=1}^{\lfloor \frac{k_i}{2} \rfloor} 2j\delta_1(j)$, $E_1 \leq 2^i E_i$ and $k_i = \frac{k_1}{2^{i-1}}$ (instead of the correct expressions: $E_i = \sum_{j=1}^{\lfloor \frac{k_1}{2} \rfloor} 2j\delta_i(j)$, $E_1 \geq 2^{i-1}E_i$ and $k_i = 2^{i-1}k_1$) in [16], its author most importantly claims

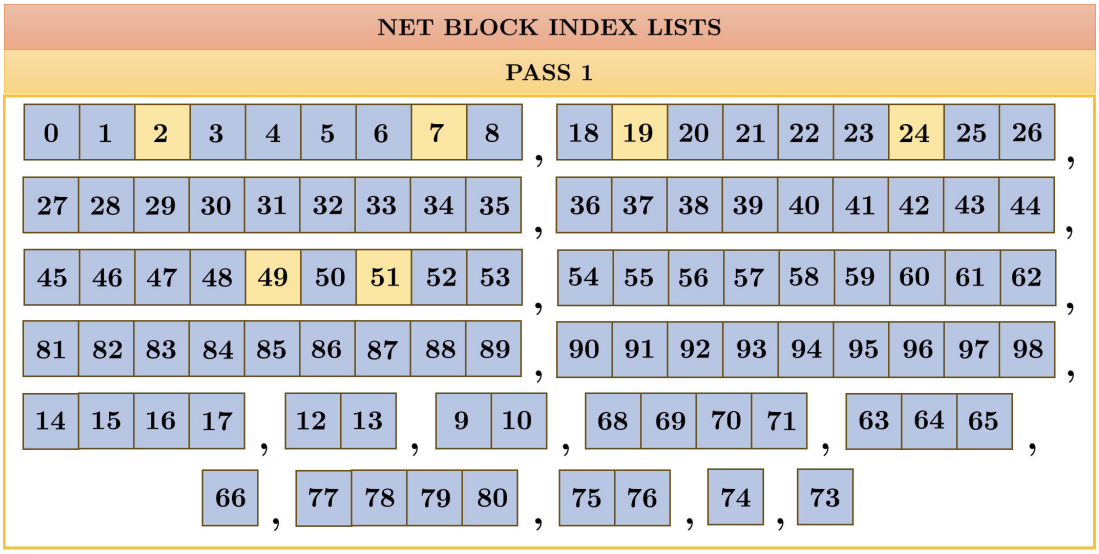


FIGURE 7. 'net_block_index_lists' dictionary at the end of Pass 1 of the cascade protocol [Undetected error locations are indicated by the colour yellow].

BLOCK NUMBER	BLOCK	PARITY MISMATCH
1	26 12 6 62 2 38 20 73 60 29 3 40 8 89 74 83 13 44	✓
2	54 24 55 97 0 56 39 11 79 57 27 23 19 58 36 98 71 88	✗
3	90 42 35 78 10 1 70 61 43 59 96 9 95 69 34 18 77 51	✓
4	72 31 99 64 17 48 25 37 81 21 7 80 45 65 94 50 28 49	✗
5	85 5 33 93 32 76 16 67 52 68 15 46 92 41 14 75 4 86	✗
6	91 53 84 82 66 87 47 30 22 63	✗

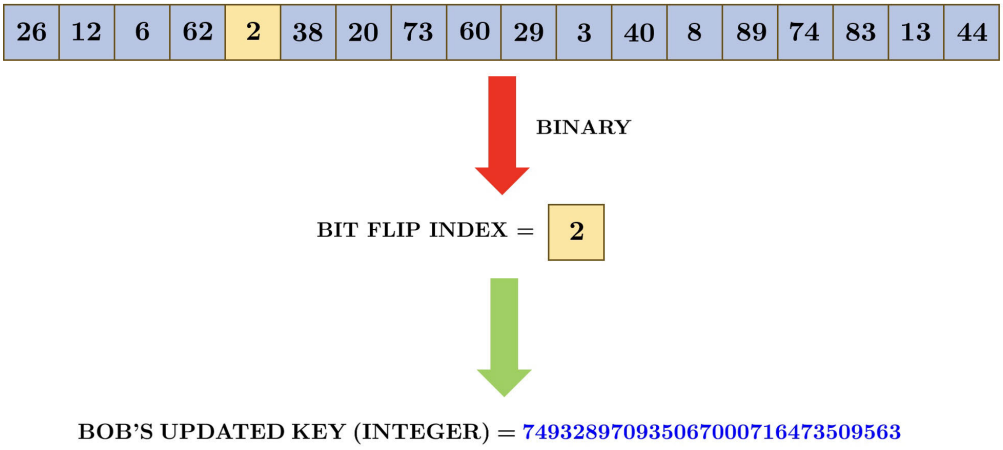
FIGURE 8. Blocks, their corresponding block numbers and error parities at the start of Pass 2 of the cascade protocol [Undetected error locations are indicated by the colour yellow].

that the inequality $\sum_{l=j+1}^{\lfloor \frac{k_1}{2} \rfloor} \delta_i(l) \leq \frac{1}{4}\delta_i(j)$ has been mistakenly assumed to be a necessary requirement for only $i = 1$ in [10]. According to the author, it's only when it is assumed to be true $\forall\, i > 1$ as well that it can be correctly said that there will be an exponential reduction in $\delta_i(j)$ $(\forall\, i)$ with the passes of Cascade. The author further states that it's not possible to set up an induction to prove that the said inequality holds true $\forall\, i > 1$ given that it does hold true for $i = 1$ since the expression involved is not a strict equality. Additionally, in [16], the inequality $E_i \leq \frac{E_{i-1}}{2}$ is presented as an assumption to arrive at the required result while this was not assumed and rather stated to follow from previously mentioned expressions and relevant calculations in [10]. The author argues that for $p = 1$ (for instance), every bit in every block of Bob's sifted key would be erroneous and consequently, any given block would always possess an even error parity leading to no error being corrected at all. In such a case, $E_i = E_{i-1}\ \forall\, i > 1$.

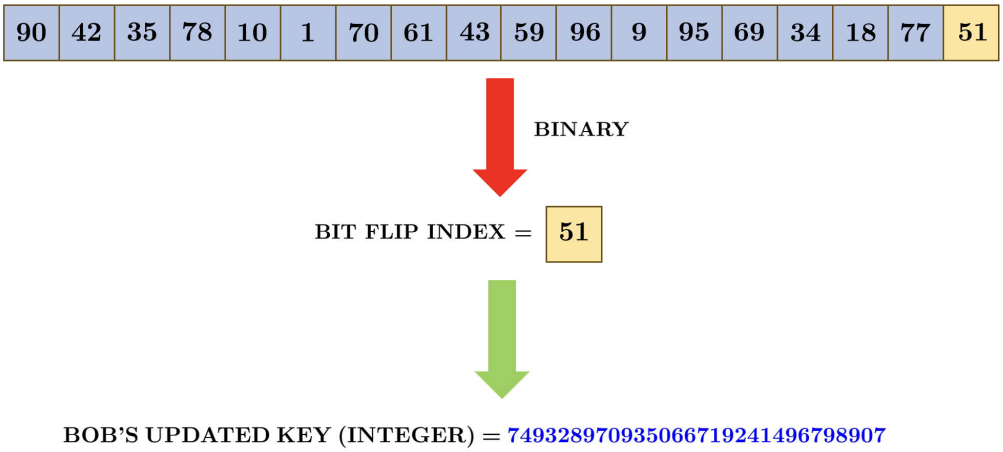
While we do agree with the author of [16] that it's not possible to set up an induction by just using the inequality $\sum_{l=j+1}^{\lfloor \frac{k_1}{2} \rfloor} \delta_1(l) \leq \frac{1}{4}\delta_1(j)$, we have systematically shown how to arrive at the desired results, namely, $\delta_i(j) \leq \frac{\delta_1(j)}{2^{i-1}}$ and $E_i \leq \frac{E_1}{2^{i-1}}$ starting with just the said inequality and the condition $E_1 \leq -\frac{\ln(\frac{1}{2})}{2}$.

We have deviated from the original proof [10] as well while doing so. While it's true that it's impossible to arrive at the inequality $\delta_i(j) \leq \frac{1}{4}\delta_{i-1}(j) + \delta_{i-1}(j)\left(1 - e^{-2^{i-1}E_{i-1}}\right)^2$ by only assuming $\sum_{l=j+1}^{\lfloor \frac{k_1}{2} \rfloor} \delta_1(l) \leq \frac{1}{4}\delta_1(j)$ to be true, clearly, it's unnecessary to even prove that this inequality holds true. In our proof, we have successfully shown that $\delta_i(j) \leq \frac{\delta_1(j)}{2^{i-1}}$ and $E_i \leq \frac{E_1}{2^{i-1}}$ which are stronger upper bounds than $\frac{\delta_{i-1}(j)}{2}$ and $\frac{E_{i-1}}{2}$ (respectively) and which clearly reveal the exponential nature of the reduction in the error probability in Cascade.

As far as the invalidity of $E_i \leq \frac{E_{i-1}}{2}$ or, $E_i \leq \frac{E_1}{2^{i-1}}$ for $p = 1$ is concerned, we point out that $p = 1$ is clearly, a pathological case, since Alice and Bob would abort the



(a) Block 1



(b) Block 3

FIGURE 9. Detection and correction of errors using BINARY in Stage 1 of Pass 2 of the cascade protocol [Undetected error locations are indicated by the colour yellow].

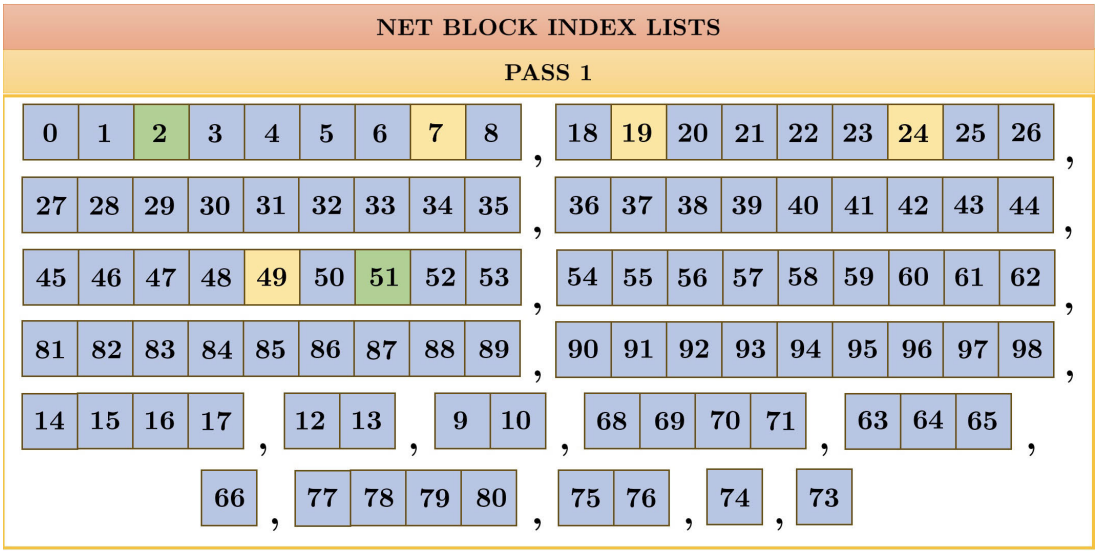
QKD protocol midway before even starting the Cascade protocol considering that the number of bit errors detected by them (in that half of the sifted key which is used for estimating the Quantum Bit Error Rate (QBER) [$= p$]) would clearly exceed any given reliable threshold on the QBER. Further, in any given case, as has already been mentioned in [10], Cascade is an optimal reconciliation protocol only as long as $p \leq 0.15$. Thus, only those values of p which are less than 0.15 are relevant to the discussion at hand.

Our probabilistic analysis thus, overcomes all the shortcomings of the analyses/proofs given in [10] and [16]. Further, we have also extensively discussed the math and physical interpretation behind the expressions for E_1 and the upper bound on $I(\omega)$ given in [10] which was not at all touched upon in [16] and to the best of our knowledge, has not been explained in detail in any related study available in the literature till now.

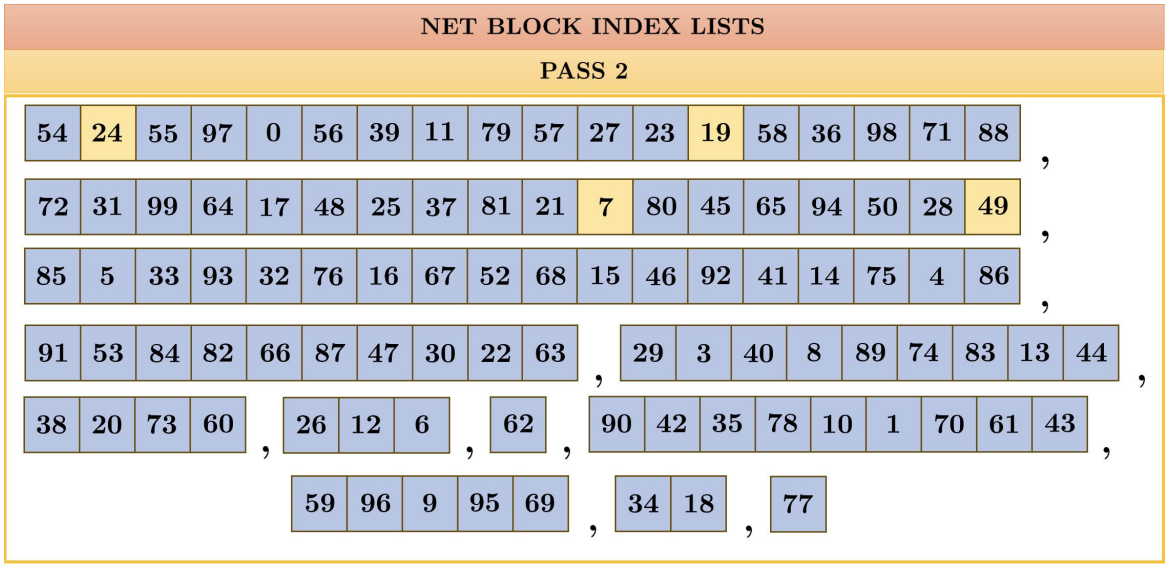
V. CASCADE IN ACTION

In this section, we give an example to illustrate how the Cascade protocol (Algorithm 3) works in practice by detailing the steps involved in the error correction of the *sifted* key string possessed by Bob. The step-by-step results shown in this example were obtained by an actual simulation of the Cascade protocol using Algorithm 3 written in Python 3.9. The notation used in this section follows from Table 1.

For the purpose of this example, we consider the QKD system's QBER to be 7.5%. Further, as shown in Fig. 3(a) and Fig. 4(a), we consider that both Alice and Bob initially possess *sifted* key strings with a length of 100 bits each. As can be discerned from Fig. 3 and Fig. 4, Bob's *sifted* key string differs from that of Alice's in 10 bits. The bits located at the indices 2, 7, 11, 19, 24, 49, 51, 67, 72 and 99 in Bob's *sifted* key string are erroneous.



(a) Blocks corresponding to Pass 1



(b) Blocks corresponding to Pass 2

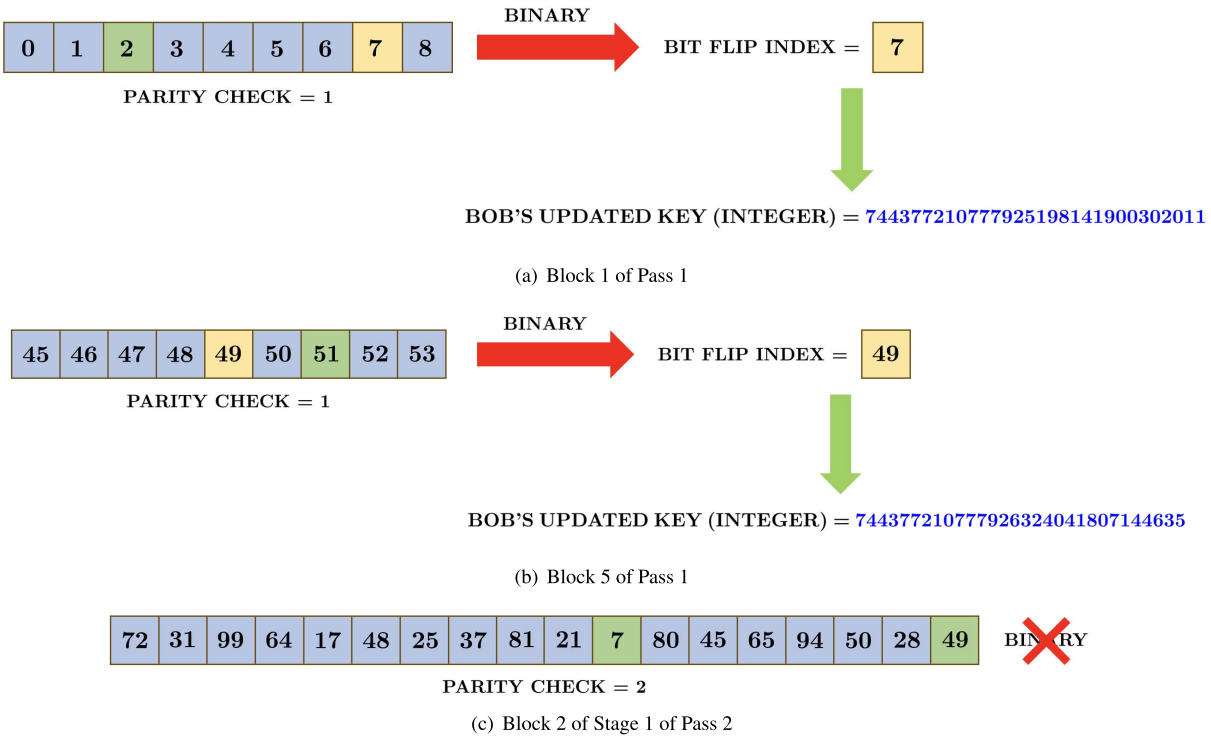
FIGURE 10. 'net_block_index_lists' dictionary after the completion of Stage 1 of Pass 2 of the cascade protocol [Undetected error locations are indicated by the colour yellow; error locations that have just been detected and corrected are indicated by the colour green].

Note that we have also explicitly shown the integer equivalents of Alice's and Bob's *sifted* key strings in Fig. 3(b) and Fig. 4(b) respectively since Cascade (as per Algorithm 3) operates on them rather than the long bit strings directly while performing error correction (see Section II for the reasons pertaining to the same).

Let us now proceed to apply the Cascade protocol. Since $p = 0.075$ and we know that $k_1 = \left\lfloor \frac{0.73}{p} \right\rfloor$ (see Section III), $k_1 = \left\lfloor \frac{0.73}{0.075} \right\rfloor = 9$. Accordingly, the block size for the 1st pass is 9. Since Algorithm 3 operates on the indices of the key strings (as already discussed in Section II), all the indices from 0 to 99 are divided into as many blocks of size 9 as is possible. As is shown in Fig. 5, we thus obtain 11 blocks with a size = 9 and 1 block with a size = 1.

Moving on, since BINARY only operates on blocks with an odd error parity, i.e., those having a parity mismatch with the corresponding blocks possessed by Alice, as is clear from Fig. 5, Bob only operates on blocks 2, 8, 9 and 12 using BINARY. Accordingly, as is shown in Fig. 6, Bob is able to identify the errors at indices 11, 67, 72 and 99 and successfully correct them by flipping the corresponding bits.

The contents of the dictionary 'net_block_index_lists' (see Algorithm 3), i.e., the even error parity blocks at the end of pass 1 are shown in Fig. 7.



(a) Block 1 of Pass 1

(b) Block 5 of Pass 1

(c) Block 2 of Stage 1 of Pass 2

FIGURE 11. Detection and correction of errors using BINARY in Stage 2 of Pass 2 of the cascade protocol [Undetected error locations are indicated by the colour yellow; error locations that have just been detected and corrected are indicated by the colour green].

As is evident from Fig. 7, the errors located at the indices 2, 7, 19, 24, 49 and 51 in Bob's *sifted* key string are still left to be detected and corrected. None of these errors could be detected in the 1st pass of Cascade since their pairwise presence in the corresponding blocks rendered the blocks with an even error parity.

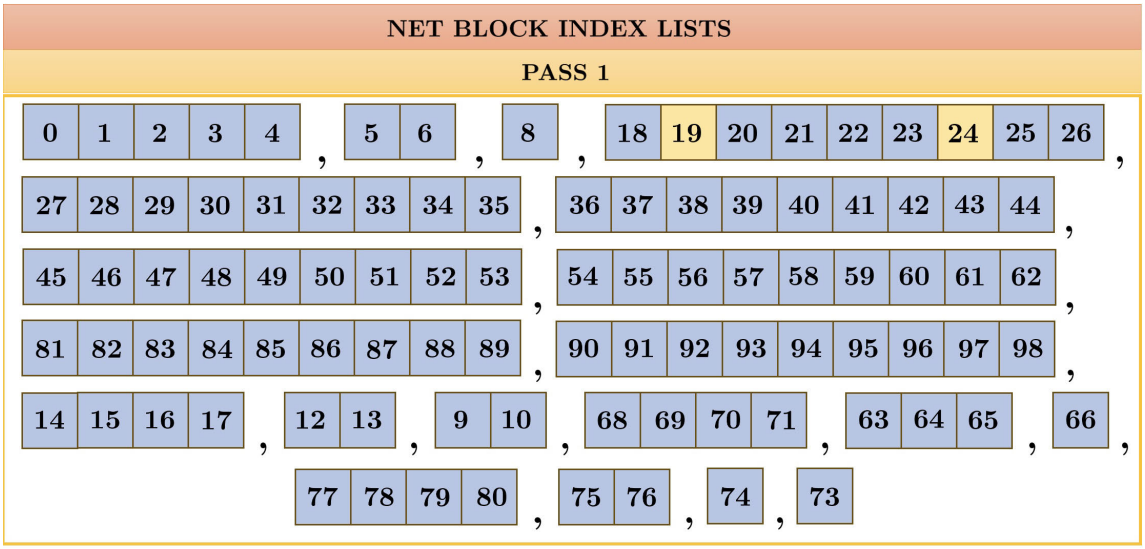
As stated earlier, from the 2nd pass onwards, random shuffling of the indices of the key string is carried out so that some of these errors may possibly be detected (and hence, corrected). Accordingly, the protocol now proceeds with the shuffling of the indices in the 2nd pass. Subsequently, since $k_2 = 2k_1 = 2 \times 9 = 18$, the shuffled net index list is divided into as many blocks of size 18 as is possible. As is shown in Fig. 8, we thus obtain 5 blocks with a size = 18 and 1 block with a size = 10.

Now, as is clear from Fig. 8, Bob only operates on (odd error parity) blocks 1 and 3 using BINARY. Accordingly, as is shown in Fig. 9, Bob is then able to identify the errors at indices 2 and 51 and successfully correct them by flipping the corresponding bits.

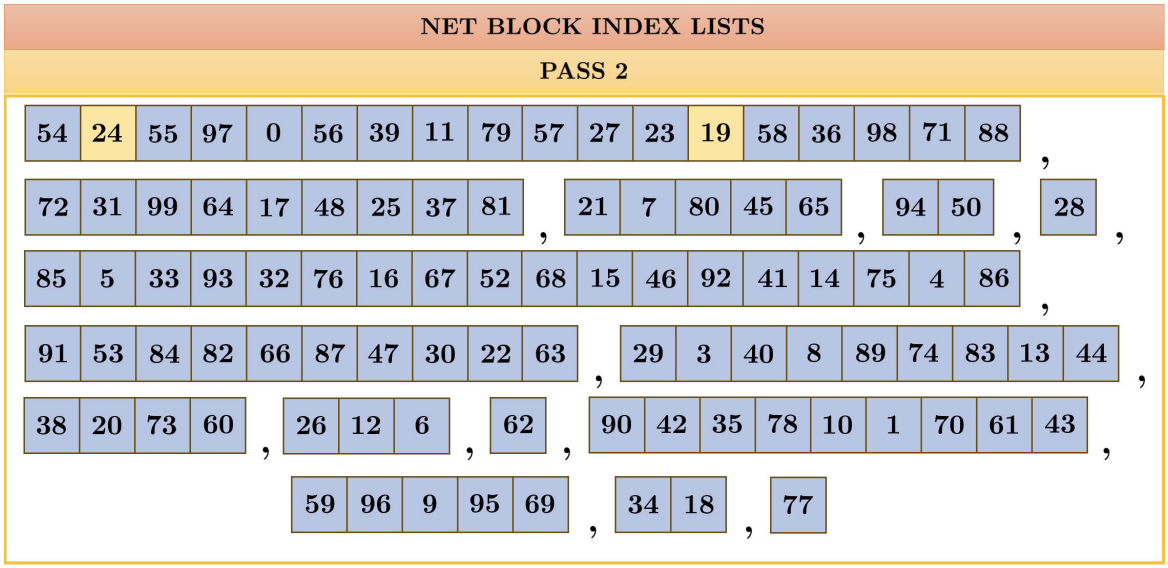
The contents of the dictionary 'net_block_index_lists', i.e., the even error parity blocks (generated in passes 1 and 2 respectively) until now, i.e., till the end of the 1st stage of the 2nd pass, are shown in Fig. 10.

Now, moving on to the 2nd stage of the 2nd pass, we find that the error locations, indices 2 and 51, that have just been detected and corrected, were also present in blocks 1 and 5 formed during the 1st pass. This can be clearly seen in Fig. 10 wherein the said indices in the aforementioned blocks have been highlighted in green. Accordingly, Bob can then backtrack to these blocks one after the other (in the ascending order of their lengths) and subsequently, use BINARY so as to find and correct errors which were hidden in the 1st pass of Cascade. However, as indicated earlier in Section II, a new error found in the 1st block (to which Bob backtracks) can *possibly* unearth an odd number of errors in the blocks formed in the 1st stage of the 2nd pass. Bob will hence be moving back and forth to *possibly* find and correct errors using BINARY. However, the important thing to note is that only those such blocks which have an odd error parity need to be operated on using BINARY and hence, the value of the parity checker (the variable 'parity_check' in Algorithm 3) needs to be checked before deciding on whether or not to operate on such a block.

In this particular instance, both the 1st and the 5th blocks formed during the 1st pass have equal lengths/sizes (= 9). For breaking this tie and hence, figuring out the block to which Bob must backtrack 1st, the blocks are arranged in the ascending order of their pass numbers and block numbers. The block with the smallest pass number and the smallest block number is considered as the 1st candidate for backtracking. Accordingly, Bob 1st operates on the 1st block formed during the 1st pass and hence, we find that since the value of 'parity_check' (=1) is odd, the block has an odd error parity. Bob can hence, operate on the block using BINARY so as to find and correct exactly 1 error. As is shown in Fig. 11, Bob is then able to find and correct the error at



(a) Blocks corresponding to Pass 1



(b) Blocks corresponding to Pass 2

FIGURE 12. 'net_block_index_lists' dictionary at the end of Pass 2 of the cascade protocol [Undetected error locations are indicated by the colour yellow].

BLOCK NUMBER	BLOCK	PARITY MISMATCH
1	81 91 71 67 17 62 12 70 53 21 32 48 13 97 84 3 99 31 38 4 20 60 47 80 5 1 7 72 68 22 98 27 33 14 15 56	✗
2	96 9 64 65 19 87 58 93 90 54 6 8 11 75 94 0 59 55 69 23 28 78 49 61 2 73 50 39 30 36 57 51 40 88 66 37	✓
3	46 34 74 89 76 10 77 95 25 44 41 63 24 16 82 35 52 85 29 45 43 83 92 26 86 18 42 79	✓

FIGURE 13. Blocks, their corresponding block numbers and error parities at the start of Pass 3 of the cascade protocol [undetected error locations are indicated by the colour yellow].

index 7 in his key. Now, this error location, index 7, was also present in the 2nd block formed during the 1st stage of the 2nd pass. Accordingly, this block also becomes a candidate for consideration for error detection and correction using BINARY. Now, since the 5th block formed during the 1st pass (length = 9) is smaller than the 2nd block formed during the 1st stage of the 2nd pass (length = 18), it becomes the next block to which Bob must backtrack. As shown in Fig. 11(b), since the value of 'parity_check' (= 1) is again odd, this block also has an odd error parity and hence, Bob is able to find and correct the error at index 49 in his key using BINARY. Now, this error location, index 49, was also present in the 2nd block formed during the 1st stage of the 2nd pass. Note that this block clearly coincides with the block identified earlier, i.e.,

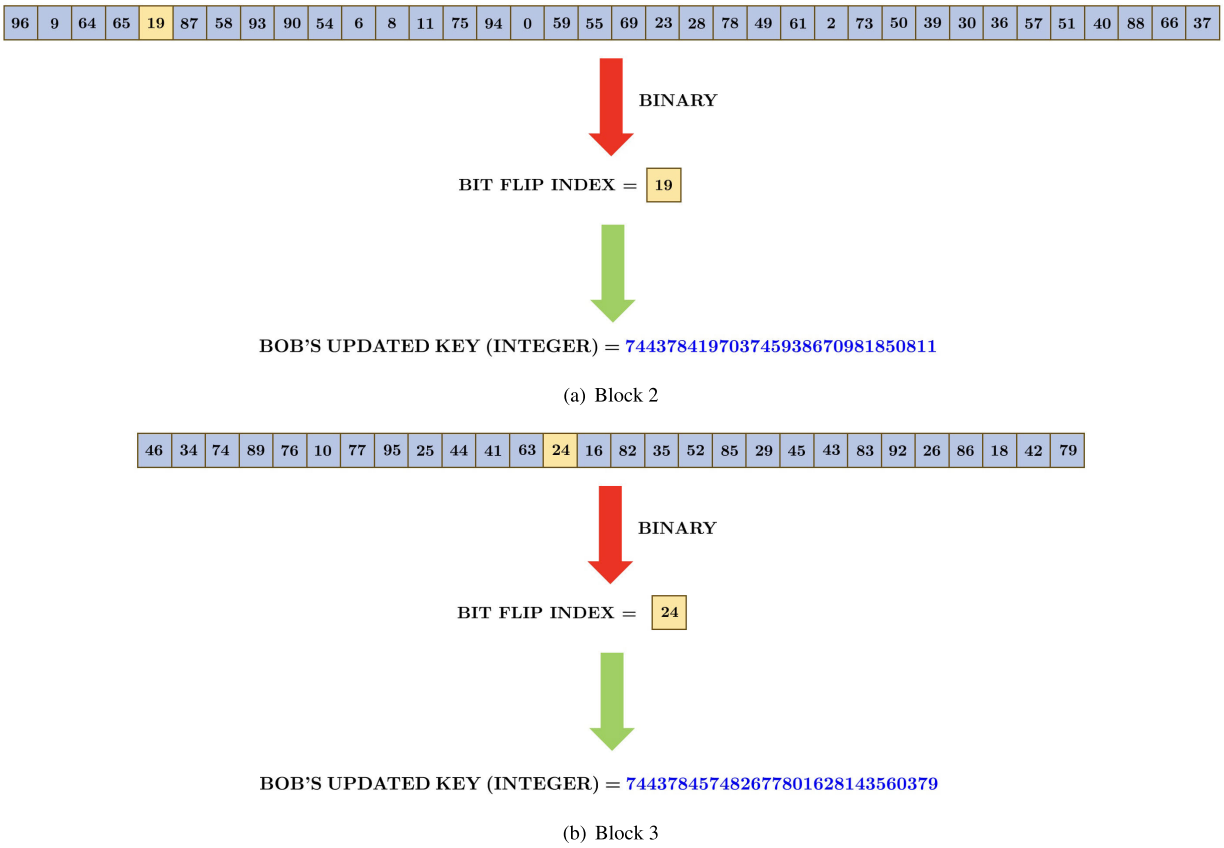


FIGURE 14. Detection and correction of errors using BINARY in Stage 1 of Pass 3 of the cascade protocol [undetected error locations are indicated by the colour yellow].

the block which contained the index 7. Accordingly, to avoid duplication, only 1 instance of this block is to be examined now. It is evident that this block has an even error parity now considering that the errors at both the indices 7 and 49 have been successfully detected and corrected now. It is therefore futile to use BINARY on this block. Bob is able to discern this by examining the value of 'parity_check' which is equal to 2 for this block (see Fig. 11(c)) and is hence, even. Operating on this block using BINARY is hence ruled out. It is worth noting that this case brings out the usefulness of 'parity_check'. Had 'parity_check' not been there (as was the case in the original algorithm given in [10]), Bob would have ended up performing 1 step of the BINARY algorithm in vain and having unnecessarily communicated with Alice. Moreover, both Alice and Bob would have unnecessarily computed the relevant parities.

Clearly, no further backtracking is to be done now since no other errors have been detected. Consequently, all the blocks have been rendered with an even error parity. The contents of the dictionary 'net_block_index_lists' after the completion of the 2nd stage of the 2nd pass, i.e., at the end of pass 2, are shown in Fig. 12.

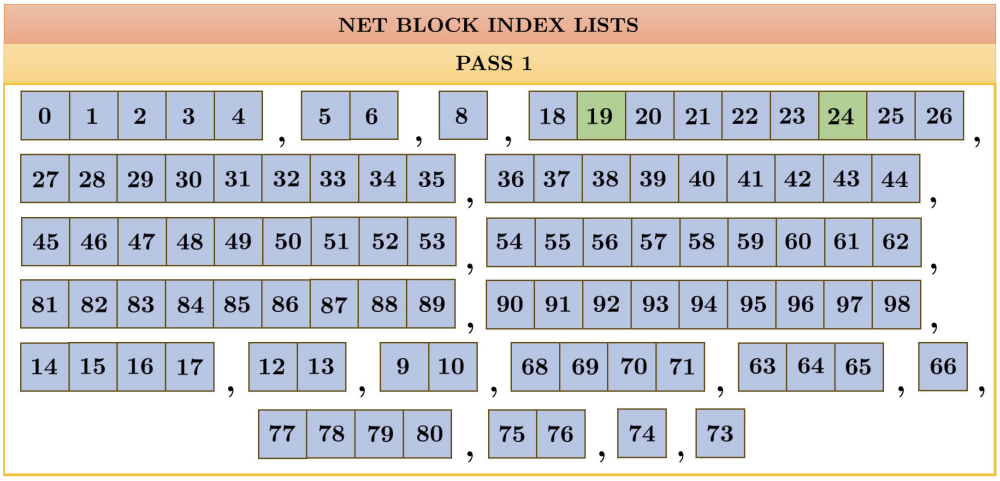
Moving on to the 3rd pass, post the shuffling of the indices and the subsequent division of the net index list into blocks of size $k_3 = 2k_2 = 2 \times 18 = 36$, Bob obtains 2 blocks with a size = 36 and 1 block with a size = 28 (see Fig. 13).

Now, as is clear from Fig. 13, Bob only operates on (odd error parity) blocks 2 and 3 using BINARY. Accordingly, as is shown in Fig. 14, Bob is then able to identify the errors at indices 19 and 24 and successfully correct them by flipping the corresponding bits.

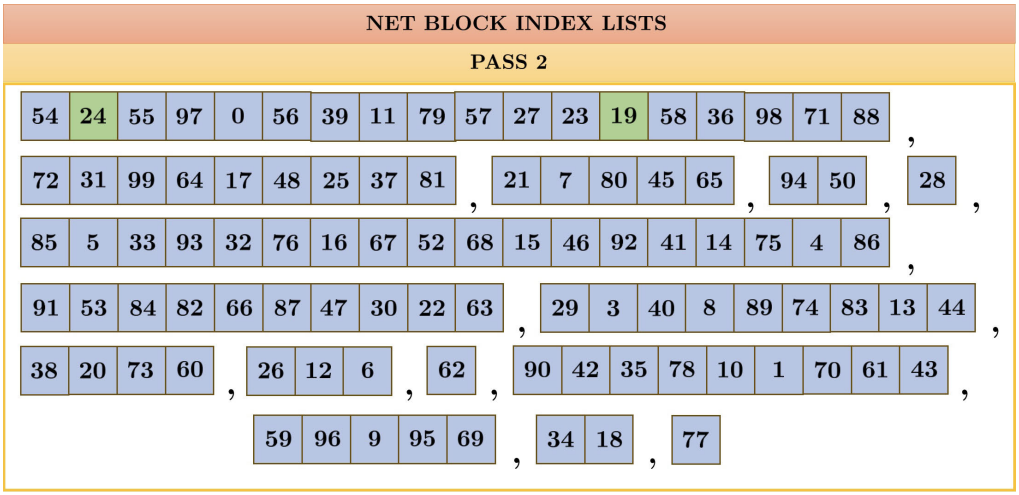
The contents of the dictionary 'net_block_index_lists', i.e., the even error parity blocks formed until now, i.e., till the end of the 1st stage of the 3rd pass, are shown in Fig. 15.

Now, moving on to the 2nd stage of the 3rd pass, we find that the error locations, indices 19 and 24, that have just been separately detected and corrected, were present together in block 4 of the 1st pass and block 1 of the 2nd pass (see Fig. 15(a) and Fig. 15(b)). Since they are present together in both the 2 blocks, the value of 'parity_check' = 2 for both the blocks and hence, these blocks are not to be operated on by Bob using BINARY (see Fig. 16).

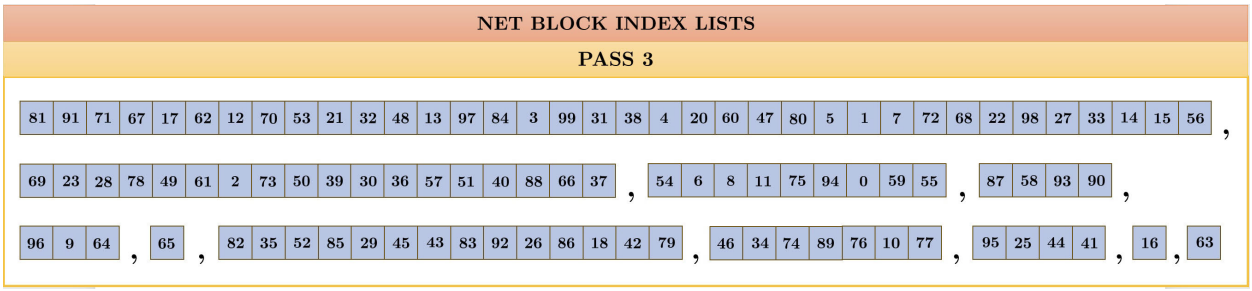
Clearly, no further backtracking is to be done and all the blocks formed until now have been rendered with an even error parity. The contents of the dictionary 'net_block_index_lists' after the completion of the 2nd stage of the 3rd pass, i.e., at the end of pass 3, are thus, the same as those after the completion of the 1st stage of the 3rd pass and



(a) Blocks corresponding to Pass 1



(b) Blocks corresponding to Pass 2



(c) Blocks corresponding to Pass 3

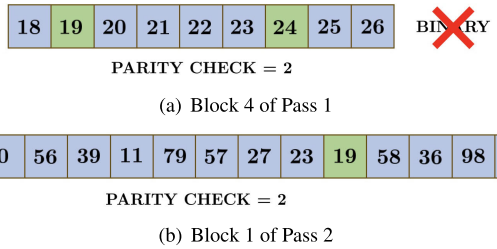
FIGURE 15. **'net_block_index_lists' dictionary after the completion of Stage 1 of Pass 3 of the cascade protocol [Error locations that have just been detected and corrected are indicated by the colour green].**

can be seen in Fig. 15 again (only the green colour doesn't hold any significance now and can be considered as blue).

Now, moving on to the 4^{th} (and final) pass, post the shuffling of the indices and the subsequent division of the net index list into blocks of size $k_4 = 2k_3 = 2 \times 36 = 72$, Bob obtains 1 block with a size $= 72$ and 1 block with a size $= 28$ (see Fig. 17).

As shown in Fig. 17, both these blocks have an even error parity and hence, BINARY is not to be used on them. This marks the end of the 4^{th} pass of Cascade and the end of the Cascade protocol.

In this particular example, all the errors present in Bob's *sifted* key string were successfully detected and corrected by the Cascade protocol. This can be easily verified by the fact that the integer equivalent of Bob's *sifted* key string after correcting the last error present at index 24 (see Fig. 14(b)) matches with that of Alice's *sifted* key string (see Fig. 3(a)).



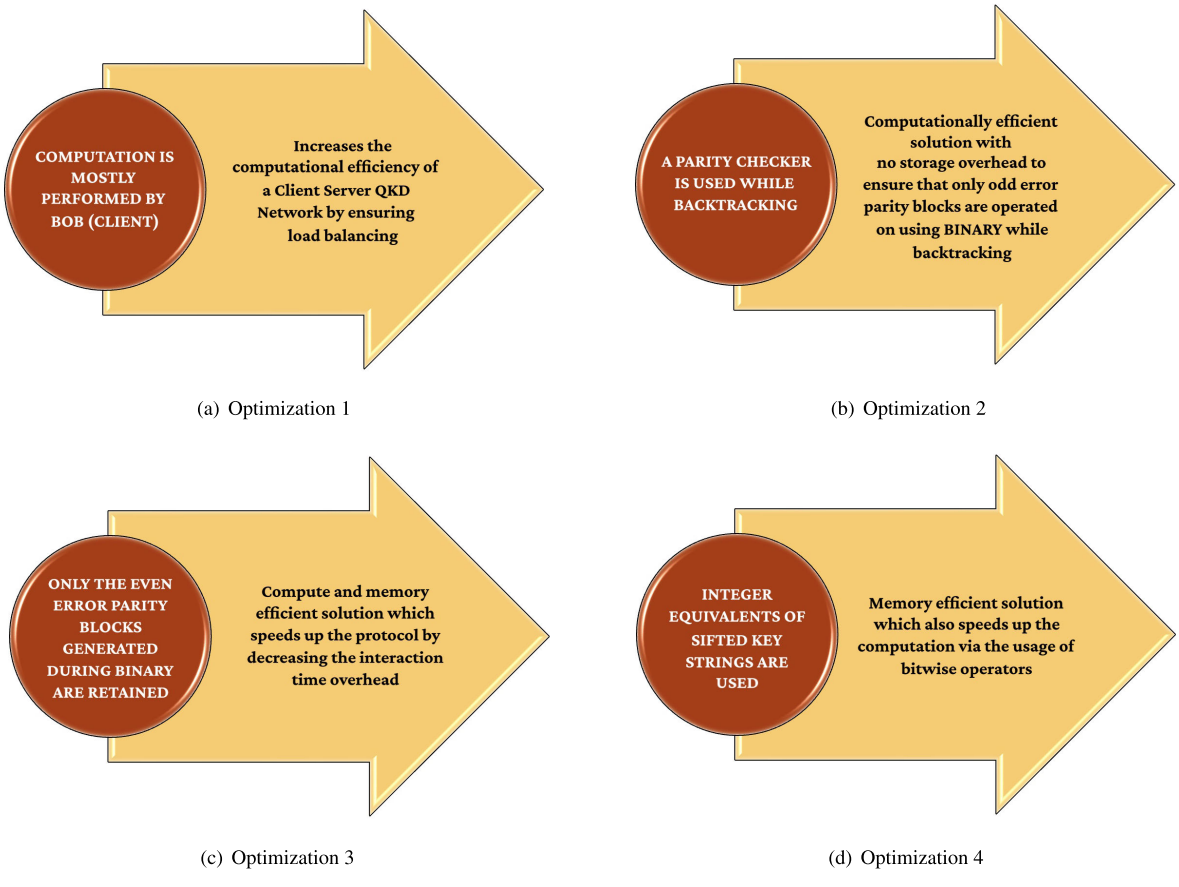
(a) Block 4 of Pass 1

(b) Block 1 of Pass 2

FIGURE 16. Stage 2 of Pass 3 of the cascade protocol [Error locations that have just been detected and corrected are indicated by the colour green].

BLOCK NUMBER	BLOCK	PARITY MISMATCH
1	65 57 12 35 11 21 2 94 67 82 58 72 47 4 42 46 9 7 78 62 92 16 14 45 95 59 38 70 73 97 91 56 25 55 34 98 51 76 53 37 90 10 33 6 29 48 1 26 27 20 40 19 22 3 24 64 8 30 41 74 77 44 81 0 88 49 32 17 80 61 5 89	✗
2	50 52 75 93 84 63 18 36 86 43 54 28 79 23 87 96 99 60 13 69 85 39 71 68 15 66 31 83	✗

FIGURE 17. Blocks, their corresponding block numbers and error parities at the start of Pass 4 of the cascade protocol.



(a) Optimization 1

(b) Optimization 2

(c) Optimization 3

(d) Optimization 4

FIGURE 18. 4 Practically useful optimizations to the cascade protocol.

It is worth mentioning that in only this particular case, all the errors present in Bob's *sifted* key string got detected and corrected in 3 passes. In general, for a given *sifted* key string possessed by Bob, the number of errors that may get finally corrected, and the number of passes in which they may get detected and corrected, depend upon the net effect of 3 major factors: the location of the errors in Bob's *sifted* key string, the value of the QBER, and the random shuffling at the start of every pass (from the 2nd pass onwards). If, for instance, there are 10 errors in Bob's *sifted* key string and all of them are present in 10 different blocks at the start of the 1st pass itself, then all those errors would get corrected in the

1st pass of the Cascade protocol. This is because each of those 10 blocks would have an odd error parity. On the other hand, if the QBER is varied, it may be possible for a larger(/lesser) number of blocks to be rendered with an odd error parity to start with and consequently, the Cascade protocol might be completed in a lesser(/larger) number of passes. Similarly, the result of the random shuffling also influences the process of error correction. Random shuffling might possibly result in a larger/lesser number of blocks with an odd error parity and this might lead to faster/slower error correction. Furthermore, in general, the way in which random shuffling influences the process of error correction is highly complex considering that even a larger number of even error parity blocks generated by it in one pass could possibly facilitate better backtracking and faster error correction in the subsequent passes. All in all, however, it can said with a high probability that with the previously mentioned choice of parameters (see Section III), the Cascade protocol does guarantee an exponentially decreasing error probability with the number of passes.

Returning back to the example in focus in this section, it can be clearly discerned from the various steps accompanied with illustrations that the Cascade protocol is simplistic yet effective. It is this very feature of Cascade that has rendered it as one of the most popular information reconciliation protocols in QKD. In fact, the world's first QKD network, the United States' Defense Advanced Research Projects Agency (DARPA) Quantum Network, successfully used the Cascade protocol for performing error correction and their team termed it as the 'best-known error correction protocol for QKD' owing to its high adaptability and efficiency [28]. Switzerland's SwissQuantum QKD Network also used the Cascade protocol and their team was able to obtain stable secret key generation rates (the number of 256-bit keys generated per day) for over a period of more than one and a half years [29]. A recent demonstration of QKD over a long distance of 421 km has also successfully utilised the Cascade protocol to obtain significantly high secret key generation rates [3]. The latest in the line of users of the Cascade protocol is a very high speed integrated QKD system which has been able to attain secret key generation rates in the kbps range over a distance of 151.5 km [30]. All these practical applications of Cascade clearly reveal that it has and continues to be a protocol of choice for error correction as a part of any given QKD protocol in QKD systems worldwide.

VI. CONCLUSION

In this study, we have thus explained the Cascade protocol in detail, backing theory with in-depth physical interpretation at every step. We have also provided an algorithmic implementation of the Cascade protocol with practically useful optimizations. Fig. 18 summarizes the 4 key optimizations proposed by us (in the circles) together with their corresponding advantages (in the arrows corresponding to the circles). Furthermore, we have extensively clarified the math behind the choice of parameter values for ensuring both: an exponential reduction in the error probability with the passes of Cascade and the minimum possible information leakage to a potential eavesdropper. Finally, we have also shown the results of a practical implementation of Cascade by providing step-by-step illustrations of Cascade in action.

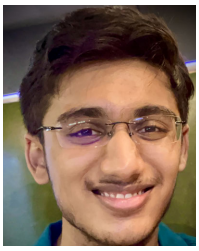
This study is sure to prove useful while developing optimized QKD systems for obtaining high computational efficiency in the information reconciliation stage with the Cascade protocol. At the same time, the in-depth physical interpretation and math behind the protocol coupled with the diagrammatic representation of Cascade in action is sure to guide academicians and people working in the industry while understanding the Cascade protocol and optimizations thereof from a practical viewpoint. To the best of our knowledge, this is the first such study on the Cascade protocol which delivers the need of the hour by providing insight driven intellect.

Future work includes a complexity analysis of our algorithm along with a timing control and synchronization enabled simulation of the Cascade protocol with reliable and accurate models of quantum hardware. This would enable us to affect a comprehensive computational efficiency cum communication latency analysis of the protocol by determining the hardware and software requirements and reconciled key generation rates. This can prove to be highly useful to the quantum communication community as it can serve as a testbed before Cascade is deployed in large and complex QKD network configurations.

REFERENCES

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. 10th ed. Cambridge, U.K.: Cambridge Univ. Press, 2010.
[2] N. Gisin, G. Ribordy, W. Tittel, and H. Zbinden, "Quantum cryptography," *Rev. Mod. Phys.*, vol. 74, no. 1, pp. 145–195, Mar. 2002.
[3] A. Boaron, G. Boso, D. Rusca, C. Vulliez, C. Autebert, M. Caloz, M. Perrenoud, G. Gras, F. Bussières, M.-J. Li, D. Nolan, A. Martin, and H. Zbinden, "Secure quantum key distribution over 421 km of optical fiber," *Phys. Rev. Lett.*, vol. 121, no. 19, Nov. 2018, Art. no. 190502.
[4] J.-P. Chen, C. Zhang, Y. Liu, C. Jiang, W. Zhang, X.-L. Hu, J.-Y. Guan, Z.-W. Yu, H. Xu, J. Lin, M.-J. Li, H. Chen, H. Li, L. You, Z. Wang, X.-B. Wang, Q. Zhang, and J.-W. Pan, "Sending-or-not-sending with independent lasers: Secure twin-field quantum key distribution over 509 km," *Phys. Rev. Lett.*, vol. 124, no. 7, Feb. 2020, Art. no. 070501.
[5] Y.-A. Chen, Q. Zhang, T. Y. Chen, W. Q. Cai, S. K. Liao, J. Zhang, K. Chen, J. Yin, J. G. Ren, Z. Chen, and S. L. Han, "An integrated space-to-ground quantum communication network over 4,600 kilometres," *Nature*, vol. 589, no. 7841, pp. 214–219, Jan. 2021.
[6] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. IT-22, no. 6, pp. 644–654, Nov. 1976.
[7] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signature and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.
[8] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484–1509, Oct. 1997.
[9] C. H. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," in *Proc. ICCCSP*, Bangalore, India, 1984, pp. 175–179.
[10] G. Brassard and L. Salvail, "Secret-key reconciliation by public discussion," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), T. Helleseth, Ed. vol. 765. Berlin, Germany: Springer, 1994.
[11] 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," *Phys. Rev. A, Gen. Phys.*, vol. 67, no. 5, May 2003, Art. no. 052303.

[12] D. Elkouss, A. Leverrier, R. Alleaume, and J. J. Boutros, "Efficient reconciliation protocol for discrete-variable quantum key distribution," in *Proc. IEEE Int. Symp. Inf. Theory*, Jul. 2009, pp. 1879–1883.
[13] D. Elkouss, J. M. Mateo, and V. Martin, "Efficient reconciliation with rate adaptive codes in quantum key distribution," *Quantum Info. Comput.*, vol. 11, nos. 3–4, pp. 226–238, 2011.
[14] T. Sugimoto and K. Yamazaki, "A study on secret key reconciliation protocol," *IEICE Trans. Fundamentals Electron. Commun. Comput. Sci.*, vol. 83, no. 10, pp. 1987–1991, Oct. 2000.
[15] H. Yan, T. Ren, X. Peng, X. Lin, W. Jiang, T. Liu, and H. Guo, "Information reconciliation protocol in quantum key distribution system," in *Proc. 4th ICNC*, Jinan, China, 2008, pp. 637–641.
[16] R. I. Y. Ng. *A Probabilistic Analysis of Binary and Cascade*. Chicago, IL, USA. Accessed: Apr. 8, 2023. [Online]. Available: http://math.uchicago.edu/~may/REU2013/REUPapers/Ng.pdf
[17] J. Martinez-Mateo, C. Pacher, M. Peev, A. Ciurana, and V. Martin, "Demystifying the information reconciliation protocol cascade," *Quantum Inf. Comput.*, vol. 15, pp. 453–477, Apr. 2015.
[18] C. Pacher, P. Grabenweger, J. Martinez-Mateo, and V. Martin, "An information reconciliation protocol for secret-key agreement with small leakage," in *Proc. IEEE ISIT*, Hong Kong, Jun. 2015, pp. 730–734.
[19] M. Toyran, M. Toyran, and S. Ozturk, "Optimized CASCADE protocol for efficient information reconciliation," *Quantum Inf. Comput.*, vol. 18, pp. 553–578, Jun. 2018.
[20] T. B. Pedersen and M. Toyran, "High performance information reconciliation for QKD with CASCADE," *Quantum Inf. Comput.*, vol. 15, pp. 419–434, Apr. 2015.
[21] A. Reis, "Quantum key distribution post processing—A study on the information reconciliation cascade protocol," M.S. thesis, School Eng., Univ. Porto, Porto, Portugal, 2019.
[22] L. Hu, H. Liu, and Y. Lin, "Parameter optimization of cascade in quantum key distribution," *Optik*, vol. 181, pp. 156–162, Mar. 2019.
[23] E. Bai, Y. Zhang, X. Jiang, Y. Wu, Z. Shi, X. Lin, and Z. Pei, "Cascade protocol with parameters and backtracking optimization," in *Proc. IEEE Int. Conf. Power Electron., Comput. Appl. (ICPECA)*, Jan. 2021, pp. 571–574.
[24] C. Crépeau. *Réconciliation Et Distillation Publiques De Secret*. Paris, France. Accessed: Apr. 8, 2023. [Online]. Available: https://www.cs.mcgill.ca/~crepeau/PS/Cre95.ps
[25] M. Mehic, M. Niemiec, H. Siljak, and M. Voznak, "Error reconciliation in quantum key distribution protocols," in *Reversible Computation: Extending Horizons of Computing* (Lecture Notes in Computer Science), vol. 12070, I. Ulidowski, I. Lanese, U. Schultz, C. Ferreira, Eds. Cham, Switzerland: Springer, Cham, 2020.
[26] B. Rijsman. *The Cascade Information Reconciliation Protocol*. Accessed: Apr. 8, 2023. [Online]. Available: https://cascade-python.readthedocs.io/en/latest/protocol.html
[27] H.-K. Mao, Q. Li, P.-L. Hao, B. Abd-El-Atty, and A. M. Iliyasu, "High performance reconciliation for practical quantum key distribution systems," *Opt. Quantum Electron.*, vol. 54, no. 3, Mar. 2022.
[28] C. Elliott, A. Colvin, D. Pearson, O. Pikalo, J. Schlafer, and H. Yeh, "Current status of the DARPA quantum network," *Proc. SPIE*, vol. 581, pp. 138–149, May 2005.
[29] D. Stucki, M. Legre, F. Buntschu, B. Clausen, N. Felber, N. Gisin, L. Henzen, P. Junod, G. Litzistorf, P. Monbaron, and L. Monat, "Long-term performance of the SwissQuantum quantum key distribution network in a field environment," *New J. Phys.*, vol. 13, no. 12, Dec. 2011, Art. no. 123001.
[30] R. Sax, A. Boaron, G. Boso, S. Atzeni, A. Crespi, F. Grunenfelder, D. Rusca, A. Al-Saadi, D. Bronzi, S. Kupijai, and H. Rhee, "High-speed integrated QKD system," *Photon. Res.*, vol. 11, no. 6, pp. 1007–1014, Jun. 2023.



ANAND CHOUDHARY (Student Member, IEEE) is currently pursuing the B.Tech. degree in engineering physics with the Indian Institute of Technology (IIT) Roorkee, India.

He is also working on the research and development of a quantum network simulator as a part of the B.Tech. degree thesis project. Foundational work on the quantum network simulator was done by him as a part of his internship with the Centre for the Development of Advanced Computing (C-DAC), Bengaluru, India. Previously, he has interned with Nanyang Technological University, Singapore, and Concentrix, India. His research interests include quantum cryptography, quantum information, quantum computation, quantum communication, artificial intelligence, deep learning, and natural language processing.

AJAY WASAN received the Ph.D. degree in coherence theory from the Indian Institute of Technology (IIT) Kanpur, India, in 1999. After that, he was a Postdoctoral Fellow of quantum optics area (both experiment and theory) with the Indian Institute of Science (IISc), Bengaluru; Kastler-Brossel Laboratory (LKB), Paris; and Kyoto University, Japan. After joining IIT Roorkee, India, as a Faculty Member, he started working on electromagnetically induced transparency (EIT) and quantum memory using Rb atoms. Currently, he is working on making multi-qubit gates using trapped neutral Rb atoms in optical tweezer arrays. He is currently a Professor at the Department of Physics, Indian Institute of Technology (IIT), Roorkee, India. This is a part of the National Mission Project on Quantum Enabled Science and Technology (QuEST) under the Department of Science and Technology (DST), Government of India. He was also involved in teaching a quantum computation and quantum information course to graduate students for last 12 years. He has over 50 refereed international publications on quantum optics. His research interests include quantum optics and quantum computation.