# A PROBABILISTIC ANALYSIS OF BINARY AND CASCADE

RUTH II-YUNG NG

Abstract. In this paper, we introduce and reconcile some of the work done on two error-correcting protocols, BINARY and CASCADE, rigorously, for a mathematically inclined audience. We also introduce our own variant of this protocol, which may be more efficient or computation-friendly, and thus a potential direction for future work. We revisit and clarify past work done in this area; in particular, we discuss proofs by Seet et al. in [5] as well as explain and critique a well-known proof by Brassard and Savail in [3]. This will be supplemented with calculations that substantiate Brassard's assumptions, and simulations of our own, to compare the protocols' efficiencies.

Contents

1. Introduction to Secret-Key Reconciliation over Public Channels 1
1.1. Parameters in Question 3
1.2. Specific Protocol in Question 3
2. Description of Protocols 4
2.1. CONFIRM 5
2.2. BINARY 5
2.3. CASCADE (Industry Standard) 6
2.4. Alternate Variant for CASCADE 8
2.5. An Example and a Clarification 8
3. Calculation of $P_i(0)$ in BINARY 10
4. Critique of Brassard and Savail's Calculations of $P_i(0)$ in CASCADE 11
4.1. Description of Brassard's Proof 11
4.2. Inconsistencies and Inaccuracies 14
5. Possible Extensions 15
Acknowledgments 16
References 16
Appendix A. Consistency of Brassard's Assumptions with Code-Runs 17
A.1. Testing $E_1 \leq -\frac{\ln \frac{1}{2}}{2}$ 17
A.2. Testing $\sum_{l=j+1}^{\frac{k_i}{2}} \delta_1(l) \leq \frac{1}{4}\delta_1(j)$ 17
Appendix B. Comparison of different Practical CASCADE Variants in (2) 19

## 1. Introduction to Secret-Key Reconciliation over Public Channels

In this section, we will introduce a problem that is common in cryptography: two parties have shared a key and would like to ensure that it is accurate through

*Date*: DEADLINES: Draft AUGUST 19 and Final version AUGUST 30, 2013.

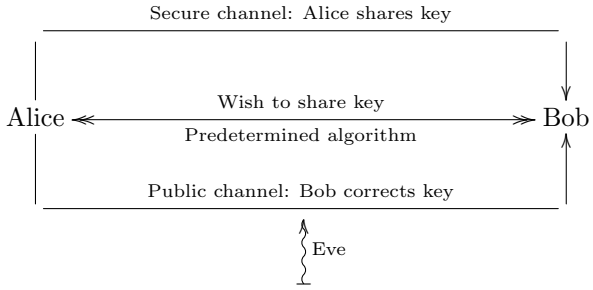


FIGURE 1. Diagrammatic Representation of Secret-Key Reconciliation.

a procedure that has to occur via a channel prone to eavesdropping by a third party. We will begin by defining the common terminology and stating the explicit problem.

**Definition 1.1.** A **key** is a finite sequence of numbers, or *bits*, where each bit is either 1 or 0. The *key-length* is the length of such a sequence, usually denoted by $n$.

Secret-Key Reconciliation involves two parties, conventionally named Alice and Bob, who wish to share a secure key.

**Definition 1.2.** A key, $K$, of length $n$ is considered *secure* if, to any party that is not Alice or Bob, $K$ is equally likely to be any key of length $n$.

Notice that every subset of a secure key is secure, even under permutation.

**Definition 1.3.** We say two keys $K = \{a_1 \dots a_n\}$ and $K' = \{a'_1 \dots a'_n\}$ are equivalent if $a_i = a'_i$ for all $i = 1, 2 \dots n$.

Alice shares a key with Bob via a Key-Distribution Algorithm, which provides each of them secure keys, which are not necessarily equivalent, but of equal length. The only way for Bob to correct his key is through a public channel, which is visible to any eavesdropper, conventionally named Eve. This is illustrated in Figure 1. Hence, a Secret-Key Reconciliation protocol aims allow Alice and Bob to end up with identical, secure keys.

**Definition 1.4.** A bit of a key $K = \{a_1, a_2 \dots a_n\}$ is considered **"leaked"** if Eve has gained information about $K$ such that she knows that there are only $2^{n-1}$ possible keys that could be $K$. However, there exists a bit $a_i \in K$ such that $K' = \{a_1, a_2 \dots a_{i-1}, a_{i+1} \dots a_n\}$ is secure.

We say that $m$ bits are leaked if $m$ separate pieces of information have the above property.

*Remark* 1.5. Notice that if $m$ bits of a key $K = \{a_1, a_2 \dots a_n\}$ are leaked, there is necessarily a key of length $n - m$ that is secure. However, this is not a sufficient condition.

**Examples 1.6.** Alice has a key $K = \{a_1, a_2 \dots a_n\}$ she wishes to reconcile with Bob. Consider the following possibilities:

- Suppose Alice attempts to send her entire key over the public channel for Bob to check against. Bob would then have a perfectly accurate key, however, $n$ bits would have been leaked and the key-length of the remaining secure key is 0.
- Suppose Alice split her key into two sets and decided to send the sum of all the bits of the key modulo 2 of each set to Bob, namely sending $\left(\sum_{i=1}^{\frac{n}{2}} a_i \mod 2\right)$ and $\left(\sum_{i=\frac{n}{2}+1}^{n} a_i \mod 2\right)$. Two bits have been leaked because removing $a_1$ and $a_n$ would be sufficient to keep the key secure.
- Suppose Alice sent $\left(\sum_{i=1}^{\frac{n}{2}} a_i \mod 2\right)$ and $\left(\sum_{i=1}^{\frac{n}{2}} (a_i + 1) \mod 2\right)$. Two bits have been leaked, however, trimming just $a_1$ is sufficient to keep the key secure. This is because the information lost to Eve was pertaining to the same subset of bits.

In the protocols studied below, it is necessary that we leak bits. Therefore, the protocols also include algorithms to remove required key bits to retain key security. Such a protocol is made up of pre-established "passes". A *pass* is an algorithm that is easily applied recursively to correct errors in the key. On the $i^{th}$ pass, the number of remaining errors in Bob's key are less than or equal to the number of errors in the $(i-1)^{th}$ pass. We study two protocols discussed in [3] `BINARY` and `CASCADE`.

1.1. **Parameters in Question.** The focus of this paper is to calculate the following:

$$P_i(j) = \text{Probability that on the } i^{th} \text{ pass, there are } j \text{ errors}$$

$$L_i = \text{Expected number of leaked bits on the } i^{th} \text{ pass}$$

For a given $i$, we wish to maximize $P_i(0)$, but not at the cost of too high a value of $\mathbb{E}(\sum L_i)$.

1.2. **Specific Protocol in Question.** We will be concerned with the Secret-Key Reconciliation between Alice and Bob for the protocols `BINARY` and `CASCADE`, which are both described in [3]. The Key-Distribution Algorithm, though irrelevant to our analysis, is intended to be Quantum Key Distribution, which makes use of orthogonal basis of quantum states to share a secret key. The specific protocol is commonly known as BB84 and is discussed in depth in [4].

The keys from Quantum Key Distribution are often used as master keys which may be used to encrypt thousands of military-use session keys, which each, in turn, is used to encrypt thousands of messages. Hence, it is important to ensure that both parties have equivalent keys. Thus, we want a high $P_i(0)$ before we stop the protocol. At the same time, the use of a quantum channel is very expensive and so we want to minimize $\mathbb{E}(\sum L_i)$ so that the bits communicated are efficiently used. This optimization motivates the research in the paper, hoping to accurately model $P_i(0)$ and $\mathbb{E}(\sum L_i)$.

## 2. Description of Protocols

Each pass of the `BINARY` protocol involves segmenting the key into blocks of equal size and calculating the sum of bits in these blocks modulo 2. Using this as a check-sum, Alice and Bob can evaluate, with a level of certainty that can be discretely determined, whether their keys are identical. They can also further segment these blocks to locate the position of particular errors. Between passes, bits are permuted. `CASCADE` is a more efficient extension of this that introduces "back-tracking", as discussed later in this paper. We begin by introducing the necessary notations that will be used throughout the paper.

**Notations 2.1.**

$n =$ Key-length

$i =$ Number of passes completed

$k_i =$ Block size on the $i^{th}$ pass

$p =$ Probability that any one bit is an error

$\sigma_i =$ Known arbitrary permutation function of the bits in a key for the $i^{th}$ pass

$B_x^i = \{a_{(x,1)}, a_{(x,2)} \ldots, a_{(x,k_i)}\}$
a subset of Bob's key where $B_x^i$ is the $x^{th}$ block in the $i^{th}$ pass and each $a_{(x,y)}$ is the bit such that $a_{\sigma_i^{-1}((x-1)k_i+y)} = a_{(x,y)}$. In other words, $a_{(x,y)}$ is the $y^{th}$ bit in the $x^{th}$ block of the $i^{th}$ pass, after applying the permutation $\sigma_i$

$A_x^i = \{a'_{(x,1)}, a'_{(x,2)} \ldots, a'_{(x,k_i)}\}$
as in $B_x^i$, but for Alice's key

$\mathcal{B} = \{a_1, \ldots, a_n\}$
Bob's entire key

$\mathcal{A} = \{a'_1, \ldots, a'_n\}$
Alice's entire key

**Definitions 2.2.** The **parity** of a block is the sum of its bits modulo two. Thus, for block $B_x^i = \{a_{(x,1)}, a_{(x,2)} \ldots, a_{(x,k_i)}\}$, the parity is given by $\left(\left(\sum_{i=1}^{k_1} a_i\right) \mod 2\right)$. If

$$\left(\left(\sum_{i=1}^{k_1} a_i\right) \mod 2\right) = 1,$$

we say that $B_x^i$ is of *odd* parity and if

$$\left(\left(\sum_{i=1}^{k_1} a_i\right) \mod 2\right) = 0,$$

we say that $B_x^i$ is of *even* parity. We say two block are of *equal* parity if they are either both odd or both even.

*Remark* 2.3. Notice that when two blocks have unequal parity, their keys differ in an odd number of positions. Conversely, when two blocks have the same parity, an even number of positions in the key differ. This means that if a block of Bob's key has unequal parity to Alice's, there must exist at least one error in the block.

However, if Bob's key has the same parity as Alice's, we cannot confirm that there are no errors in the block as there may be an even number of errors in it.

Finally, we assume that $n$ and $k_i$ are both a power of 2, as required by the protocol. In practice, this can be achieved by truncating the key to a power of 2, or padding the key with bits of known value (such as make them all 0) such that the key-length reaches the next highest power of 2. These bits are clearly removed at the end of the protocol.

2.1. **CONFIRM.** Although it is not a protocol in itself, CONFIRM is a method described in [2], which corrects exactly one bit from a block at a time. Consider a block $\{a_{(x,1)}, a_{(x,2)} \ldots, a_{(x,k_i)}\}$ in Bob's key that is of different parity to Alice's block $\{a'_{(x,1)}, a'_{(x,2)} \ldots, a'_{(x,k_i)}\}$. Assume that both of these parities are known.

When $k_i = 1$, this is trivial. When $k_i = 2$, Alice sends Bob $a'_{(x,1)}$ and Bob can easily determine which bit is erroneous. This results in the leakage of 1 bit. Similarly, given a block $\{a_{(x,1)}, a_{(x,2)} \ldots, a_{(x,k_i)}\}$, which Bob knows to be of different parity than $\{a'_{(x,1)}, a'_{(x,2)} \ldots, a'_{(x,k_i)}\}$, Alice can send Bob $\left(\left(\sum_{j=1}^{\frac{k_i}{2}} a'_{(x,j)}\right) \mod 2\right)$ and Bob can check this against the value of $\left(\left(\sum_{j=1}^{\frac{k_i}{2}} a_{(x,j)}\right) \mod 2\right)$. If these are of equal parity, Bob knows the error is in the latter half of the key and can create a block of size $\frac{k_i}{2}$ which he knows contains at least one error. Using this method recursively, we know that we can always find one error between two keys known to be of unequal parity after leaking $\log_2 k_i$ bits, since this is exactly the number of times a block (known to be a power of 2) can be split in half until we are left with a single bit.

2.2. **BINARY.** The BINARY protocol follows from CONFIRM in the following steps, as in described in [3]:

(Step 1) Alice selects a subset of bits in her key to share over the public channel. Bob can estimate $p$ from this. Note that the subset of bits should be randomly chosen across the key to get an even spread of errors (avoiding random and systematic errors of the key-distribution protocol). Both Alice and Bob remove these bits from both their keys and continue the protocol with remaining key, possibly truncating it further such that $n$ is a power of 2.

(Step 2) Alice and Bob apply a pre-determined (and available to Eve) permutation $\sigma_1$ to their keys to evenly distribute the errors across the key. This is proposed by [4] to make sure the errors in the keys are identically and independently distributed before error-correction begins. This makes the protocol more efficient because there will be more blocks with 1 error in the first pass. The key from this point onward is taken to be the original key under the permutation $\sigma_1$. The rest of the protocol proceeds in passes, renaming each $\sigma_i(a_i), \sigma_i(a'_i)$ as $a_i, a'_i$ respectively. Starting with $i = 1$, $i$ can be incremented so long as $k_i < n/2$. These keys are denoted $\mathcal{A} = \{a'_1, a'_2 \ldots, a'_n\}$, and $\mathcal{B} = \{a_1, a_2 \ldots, a_n\}$, as described in Notations 2.1.

(Step 3) Each pass $i$ is determined as follows:

(i) When $i = 1$, we calculate $k_1$, based on $p$, to be the initial block size. This optimal $k_1$ is both a power of 2 and results in the highest expected number of blocks containing 1 error. Essentially, we assume this to be the power of 2 closest to $\frac{1}{p}$. When $i \neq 1$, take $k_i = 2k_{i-1}$.
(ii) Alice and Bob each define blocks $A_x^i$ and $B_x^i$, as defined in 2.1 by splitting their keys into blocks of size $k_i$. Specifically, for $x = 1, 2 \ldots \frac{n}{k_i}$ where we have $a_{(x,y)} \in B_x^i$ and $a'_{(x,y)} \in A_x^i$ such that for $i = 1$, $a_{(x,y)} = a_{(x-1)k_i+y}$ and $a'_{(x,y)} = a'_{(x-1)k_i+y}$. And for $i \neq 1$, we have $a_{(x,y)} = a_{\sigma_i((x-1)k_i+y)}$ and $a'_{(x,y)} = a'_{\sigma_i((x-1)k_i+y)}$.
(iii) Alice sends Bob the parity of each $A_x^i$, which Bob checks against the parity of each $B_x^i$. When this parity is unequal, they perform `CONFIRM` on these blocks to find and correct one error in the original key.
(iv) If necessary, we proceed to pass $i + 1$ as long as $k_i \leq \frac{n}{4}$.

Clearly, so long as the condition $k_i \leq \frac{n}{4}$ is met, we can always proceed to the $i + 1$ pass from the $i$ pass. Therefore, Alice and Bob should optimize the number of passes needed to have $P_i(0)$ as high as they require, while minimizing $\mathbb{E}(\sum L_i)$. This need for optimization motivates our calculations.

2.3. `CASCADE` **(Industry Standard).** To improve the efficient use of the key-length (i.e., minimizing $\mathbb{E}(\sum L_i)$), Brassard suggests the following modification of `BINARY` in [3], which he terms `CASCADE`. In a nutshell, `CASCADE` is an improvement on `BINARY` that allows errors corrected on the $i^{th}$ pass to reveal the location of additional errors under permutations of the key in some pass $i'$, where $i' \neq i$. This reduces the number of passes that need to be done, and minimizes $\mathbb{E}(\sum L_i)$. This is discussed in detail below.

(Step 1) (As in Step 1 of `BINARY`)
(Step 2) (As in Step 2 of `BINARY`)
(Step 3) Pass i:
(i) For $i = 1$, the entire pass proceeds as in `BINARY`. For $i \neq 1$, we begin as in a pass of `BINARY`. First create an empty set $S$ which will contain all blocks that are known to contain an error which has not been corrected. For each error $e \in B_x^i$ that we find and correct, remove this block from $S$. We can use the permutation $\sigma_i^{-1}$ to retrieve $e = a_j$, the position of $e$ in Bob's original key, since $e \in \mathcal{B}$ (the error must be one of the bits in Bob's key). Then, notice that $a_j = a_{(x,j')} \in B_x^{i'}$ for some $x, j'$ in each of $i' = 1, 2, \ldots, (i-1)$. Then, for each of the $i-1$ blocks $B_x^{i'}$ identified above, if $B_x^{i'} \notin S$, we insert it into the group and if $B_x^{i'} \in S$, we take it out.
Remark: Note that this set $S$ can be seen as the set of all blocks which currently have an odd number of errors. Each time an error is corrected in some pass, correcting the error across the other passes will change the parity of one of Bob's blocks in each other pass. This means that either a block assumed to have an odd number of errors now has an even number of errors, or a block that has an even number of errors now has an odd number of errors. Hence, after this correction, the set $S$ can be modified to reflect this block

or remove an existing block in $S$. This is called "back-tracking" the error through other passes.

(ii) Now, for $i \neq 1$, we recursively choose the block of lowest block size in $S$, we perform CONFIRM on the block to arrive at an additional error. This error $e \in B_x^{i'}$ is removed from $S$ and also back-tracked through the other passes, specifically, adding or removing an element from $S$ for each of $j = 1, 2, \ldots n$, $j \neq i'$. This is continued until $S = \emptyset$. This procedure is described in detail using an example in Section 2.5.

There are two motivations for using CASCADE over BINARY. In particular, $\mathbb{E}\left(\sum L_i\right)$ is minimized in the former in the following two ways.

(1) The BINARY protocol is embedded in the CASCADE protocol in the sense that at the $i^{th}$ pass in CASCADE, we begin the protocol by first correcting the same errors as if a single pass of BINARY is performed at this point. The modification that we made shows that CASCADE corrects errors that are strictly greater in number than BINARY. Therefore, for each pass, it is preferable to perform CASCADE than BINARY. Not only is the expected number of corrected errors per pass lower in CASCADE than in BINARY, the number of passes that need to be carried out to achieve a fixed $P_i(0)$ is also lower. More blocks with even numbers of errors are evaluated contributing one bit per block to $\sum L_i$ when more passes are performed; thus, there will be less contribution by blocks with even numbers of errors to $\sum L_i$.

(2) Notice that under CASCADE, to achieve a certain $P_i(0)$ given an expected $p$, there is a fixed expected number $N$ of errors that need to be corrected. For each of these errors, they contribute $(\log_2 k_i + 1)$ bits to $\sum L_i$ when corrected, for some $k_i$ (first 1 bit in determining the parity of the block and $\log_2 k_i$ bits in CONFIRM). As above, strictly more errors are corrected in a pass of CASCADE as opposed to BINARY. This means that under BINARY, the expected $k_i$ at which the error is corrected is higher than that in CASCADE. Therefore, the odd-blocks contribute less to $\sum L_i$ as well, since $\mathbb{E}(\log_2 k_i + 1)$ is lower.

*Remark* 2.4. Notice that under CASCADE,

$$\sum L_i \neq \sum_{\text{i}} \left( \sum_{\text{even blocks}} 1 + \sum_{\text{odd blocks}} (\log_2 k_i + 1) \right),$$

because each block may contain more than one corrected error and a block that initially has an even number of errors may have errors corrected due to back-tracking. The above claim that $\sum L_i$ is always strictly reduced by using CASCADE than BINARY is based on,

$$\sum L_i = \sum_{\text{i}} \left( \sum_{\text{initially even blocks}} 1 + \sum_{\text{initially odd blocks}} (\log_2 k_i + 1) + \sum_{\text{other errors corrected}} (\log_2 k_i) \right).$$

This means that,

$$\sum L_i = \sum_{\text{i}} \left( \frac{n}{k_i} + \sum_{\text{errors corrected}} (\log_2 k_i) \right).$$

2.4. **Alternate Variant for CASCADE.** While the above version of CASCADE is the one initially intended by Brassard in [3], the protocol is somewhat differently described in papers such as [2], [4] and [1]. This led us to consider an alternate protocol in an attempt to further optimize $\mathbb{E}(\sum L_i)$ without compromising $P_i(0)$.

(Step 1) (As in Step 1 of BINARY)
(Step 2) (As in Step 2 of BINARY)
(Step 3) Pass i: For $i = 1$, the entire pass proceeds as in BINARY. For $i \neq 1$, we begin by checking the parities of all $B_x^i$ where $x = 1, 2 \ldots \frac{n}{k_i}$. For all $B_x^i$ of unequal parity to $A_x^i$, we insert $B_x^i$ into a set $S$. We then recursively choose the block of lowest-block size, $B_x^{i'} \in S$, correct an error in it, remove it from $S$ and back-track the error through all other passes (namely, adding or removing an element from $S$ for each of $j = 1, 2, \ldots n$, $j \neq i'$). This is continued until $S = \emptyset$.

The only modification is in step (3) of the protocol, where, instead of immediately performing BINARY on a pass when $i \neq 1$, we simply check the parities of the blocks and immediately form a set $S$ of blocks, which are known to contain errors. We do not solve them immediately. We then proceed as in the protocol until $S = \emptyset$ to complete the pass.

This modification begs the following question:

**Question 2.5.** *When all errors have been corrected, is $\mathbb{E}(\sum L_i)$, on average, under this method of CASCADE lower than under the industry CASCADE?*

As the simulation results in Appendix B suggest, it is unclear which of these methods is superior. One can also construct examples where either variant performs better. While our research in this area yielded no complete resolution for the above question, we present the following discussion to suggest why it is not obvious which protocol is better.

Perhaps our variant is intuitively more efficient in terms of $\mathbb{E}(\sum L_i)$ since errors are immediately addressed at a lower pass whenever possible. Therefore, one would suspect that if we ran both protocols until there were no more errors in either key, errors on average would have been addressed at lower passes under the method of CASCADE we propose. Therefore, should the number of errors corrected at each pass be comparable, under the two protocols, one would know that the variant we propose is more efficient.

However, we were not able to prove this, in part due to a property of CONFIRM. Notice that when a particular block has an odd number of errors, arranged in a specific way, one particular error will be determined by CONFIRM. However, when an error is back-tracked to a pass, it could be any error in the block. Therefore, perhaps by using a pass as in BINARY first, more errors would be generated at the $i^{th}$ pass in order to remove errors at each other pass that would not otherwise be corrected if we back-tracked at every opportunity available.

2.5. **An Example and a Clarification.** Below is an example comparing BINARY as well as the two versions of CASCADE. In Figure 2.5, the points along the line refer to the relative position of errors in the key $\mathcal{B}$ at the beginning of correcting errors and under permutations $\sigma_2, \sigma_3$, in passes $i = 1, 2, 3$. This key contains 8 errors and has 8 blocks in pass 1. Bits that do not contain errors are not reflected in the diagram.

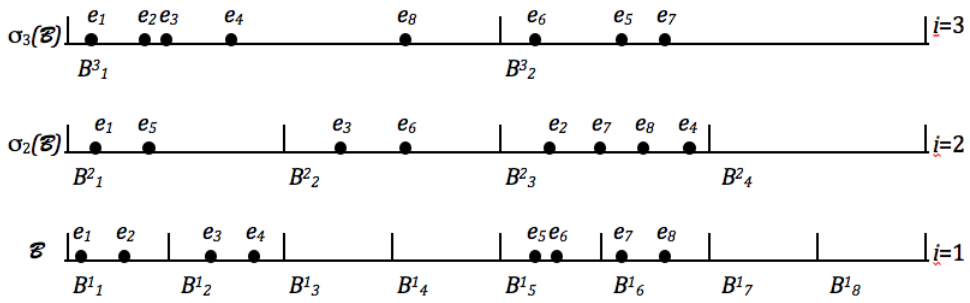


FIGURE 2. Example for Comparison of Protocols

Here is how the protocol proceeds under BINARY in the order in which the errors are corrected:

(1) Pass 1: no odd blocks found in $\mathcal{B}$
(2) Pass 2: no odd blocks found in $\sigma_2(\mathcal{B})$
(3) Pass 3: Odd blocks $B_1^3$ found. $e_8 \in B_1^3$ corrected
(4) Pass 3: Odd blocks $B_2^3$ found. $e_6 \in B_2^3$ corrected

As for the industry standard of CASCADE, we proceed as follows (for clarity, we also denoted the contents of the set $S$ at each step):

(1) Pass 1: no odd blocks found in $\mathcal{B}$
(2) Pass 2: no odd blocks found in $\sigma_2(\mathcal{B})$
(3) Pass 3: Odd blocks $B_1^3, B_2^3$ found. $e_8 \in B_1^3$ corrected. $e_6 \in B_2^3$ corrected
    $S = \{B_5^1, B_6^1, B_2^2, B_3^2\}$
(4) Pass 3: $e_5 \in B_5^1$ corrected
    $S = \{B_6^1, B_1^2, B_2^2, B_3^2, B_2^3\}$
(5) Pass 3: $e_7 \in B_6^1$ corrected
    $S = \{B_1^2, B_3^2\}$
(6) Pass 3: $e_1 \in B_1^2$ corrected
    $S = \{B_1^1, B_3^2, B_1^3\}$
(7) Pass 3: $e_2 \in B_1^1$ corrected
    $S = \emptyset$

Here is how the protocol will proceed under our proposed variant of CASCADE:

(1) Pass 1: no odd blocks found in $\mathcal{B}$
(2) Pass 2: no odd blocks found in $\sigma_2(\mathcal{B})$
(3) Pass 3: Odd blocks $B_1^3, B_2^3$ found
    $S = \{B_1^3, B_2^3\}$
(4) Pass 3: $e_8 \in B_1^3$ corrected
    $S = \{B_6^1, B_3^2, B_2^3\}$
(5) Pass 3: $e_7 \in B_6^1$ corrected
    $S = \emptyset$

At this point, we add that while the analysis of both $P_i(0)$ and $L_i$ are interesting to understanding the overall efficiency of the system, this trade-off is mostly a moot point in application. To minimize the effect of Eve possibly having partial knowledge of the key, a slight difference in the key that is used to encode or decode messages renders the messages completely unintelligible to one or both parties.

Therefore, the computation of $P_i(0)$ is extremely relevant to the functioning of the protocol, whereas the computation of $L_i$ only elucidates the system's potential cost and compares protocols to each other. Additionally, we believe the calculation of $L_i$ to follow easily from computations of $P_i(0)$, as evident in [5] with `BINARY`.

Thus, we will consider the study of $L_i$ to be outside the scope of the paper for the rest of this discussion and leave it as an extension.

## 3. Calculation of $P_i(0)$ in `BINARY`

Let $\Delta_i(j-y \mid j)$ be the probability that on the $i^{th}$ pass, $j-y$ errors are corrected given that there were $j$ errors on the $(i-1)^{th}$ pass. Recall that $P_i(j)$ refers to the probability that after the $i^{th}$ pass is completed, $j$ errors remain in the key. This section aims to create an explicit formula for $\Delta_i(j-y \mid j)$ in `BINARY` and compute $P_i(0)$ for all $i$ in `BINARY`. The ideas and theorems in this section closely follow those in [5], but with more mathematical rigor.

**Theorem 3.1.** *$P_i(y)$ may be defined recursively as follows:*

$$P_i(y) = \begin{cases} \binom{n}{y} p^y (1-p)^{n-y}, & \text{if } i = 1 \\ \\ \sum_{j=y}^{y+\frac{n}{k_i}} \left(P_{i-1}(j)\ \Delta_i(j-y \mid j)\right), & \text{if } i > 1\ . \end{cases}$$

*Proof.* Due to the protocol specifications, at most one error is corrected in each block per pass and this happens only in blocks with odd parity. Therefore, at most $\frac{n}{k_i}$ errors can be corrected at the $i^{th}$ pass.

Thus, for $i > 1$ we can consider this as $\frac{n}{k_i}$ different cases, depending on the number of errors that were corrected at this pass. We know that there were $j$ initial errors, so take the case where $j-y$ errors are corrected and $y$ errors remain, for some $y \leq \frac{n}{k_i}$. The probability of this is the chance that there are $j$ errors at the $i-1$ pass, $P_{i-1}(j)$, and that $y$ remain and $j-y$ were corrected, $\Delta_i(j-y \mid j)$. Therefore, the probability that there are $j$ initial errors and that $y$ errors remain after the $i^{th}$ pass is $P_{i-1}(j)\ \Delta_i(j-y \mid j)$. We sum over all cases of $y \leq \frac{n}{k_i}$ and arrive at $P_i(y) = \sum_{j=y}^{y+\frac{n}{k_i}} (P_{i-1}(j))$. □

**Theorem 3.2.** *The quantity $\Delta_i(j-y \mid j)$ is given by,*

$$\Delta_i(j-y \mid j) = \frac{\binom{\frac{n}{k_i}}{j-y}}{\binom{n}{j}} C_{i,j,y},$$

*where $C_{i,j,y}$ is the coefficient of $x^j$ in the expansion of:*

$$\left(\frac{(1+x)^{k_i} - (1-x)^{k_i}}{2}\right)^{j-y} \left(\frac{(1+x)^{k_i} + (1-x)^{k_i}}{2}\right)^{\frac{n}{k_i}-(j-y)}.$$

*Proof.* Notice that the event where $j-y$ errors are corrected and $y$ errors remain is equivalent to the event where $j-y$ have an odd number of errors and the remaining $\frac{n}{k_i}-(j-y)$ blocks have an even number of errors. Consider, therefore, the following polynomial with dummy variable $x$:

$$\left(\binom{k_i}{1}x + \binom{k_i}{3}x^3 + \dots \binom{k_i}{k_i-1}x^{k_i-1}\right)^{(j-y)} \left(\binom{k_i}{0} + \binom{k_i}{2}x^2 + \binom{k_i}{4}x^4 \dots \binom{k_i}{k_i}x^{k_i}\right)^{\frac{n}{k_i}-(j-y)}$$

$$= \left(\frac{(1+x)^{k_i}-(1-x)^{k_i}}{2}\right)^{j-y} \left(\frac{(1+x)^{k_i}+(1-x)^{k_i}}{2}\right)^{\frac{n}{k_i}-(j-y)}.$$

The first product term gives the number of ways to have $j-y$ blocks with an odd number of errors, and the second product term gives the number of ways we can have $\frac{n}{k_i} - (j-y)$ blocks with an even number of errors. The resulting polynomial in $x$ has the property that the coefficient of $x^j$ is the total number of ways these errors can be distributed such that $j-y$ blocks have an odd number of errors.

There are $\binom{\frac{n}{k_i}}{j-y}$ ways to arrange $j-y$ blocks with an odd number of errors amongst the $\frac{n}{k_i}$ blocks. Finally, note that there are $\binom{n}{j}$ ways to distribute $j$ errors in a key of length $n$ with no restrictions. Therefore, the probability that $j-y$ errors are solved at pass $i$ is $\binom{\frac{n}{k_i}}{j-y}C_{i,j,y}$. From this, we have that $\Delta_i(j-y \mid j) = \frac{\binom{\frac{n}{k_i}}{j-y}}{\binom{n}{j}}C_{i,j,y}$. □

Notice that these equations are sufficient to find $P_i(0)$ for some $i$ if we are also given $k_1$ and $p$. The consistency of the above calculations with simulations results is discussed in [5].

## 4. Critique of Brassard and Savail's Calculations of $P_i(0)$ in CASCADE

As the main contribution of this paper, this section critiques Brassard and Savail's proof, elaborating on the approximation that they used. Though widely used, Brassard and Savail's lower-bound calculation of $P_i(0)$ contains certain weaknesses. It has also been shown, via simulation, to be so loose a lower-bound that it approximates $P_i(0)$ in CASCADE to be lower than BINARY for the same $i$, according to [5]. This is despite the fact that BINARY subsumes CASCADE. Moreover, for small values of $n$, [6] shows, that the approximation ceases to be a lower-bound in all cases via simulation.

### 4.1. Description of Brassard's Proof.

This will include steps and details that the original proof leaves out. For ease of reference, we adopted the same notation as Brassard does, for everything that is not listed in Notations 2.1.

**Notations 4.1.**

$\delta_i(j) =$ Probability that after the $i^{th}$ pass there are $2j$ errors remaining in Bob's key

$E_i =$ Expected number of errors after the completion of pass $i$ in some $B_x^1$, a block from pass 1, updated to reflect any errors that have been found

$\gamma_i =$ Probability of correcting at least 2 errors in pass $i > 1$, in some $B_x^1$ which is known to contain errors

**Theorem 4.2.** *Right before pass 1, let random variable $X$ be the number of errors remaining in each block. Then, $X$ is a binomial random variable where,*

$$X \sim Bin(k_1, p).$$

*Additionally,*

$$\delta_1(j) = \mathbb{P}(X = 2j) + \mathbb{P}(X = 2j+1),$$

*and,*

$$E_i = \sum_{j=0}^{\frac{k_i}{2}} ((2j)(\delta_1(j)))\,.$$

*Proof.* Notice that we applied a permutation $\sigma_1$ to the bits in the key before beginning the protocol. This assures that the errors are independent and identically distributed within the key. Furthermore, we have that $p$ is the probability that each bit is an error. Therefore, for a single block of size $k_1$, we have $X \sim Bin(k_1, p)$.

After pass 1, there are two cases where $2j$ errors remain.

(1) The block originally had $2j$ errors. This being an even number of errors, we know none of them are corrected. This occurs with probability $\mathbb{P}(X = 2j)$.
(2) The block originally has $2j + 1$ errors. This being odd, exactly 1 error is corrected and the block consequently is left with $2j$ errors. This occurs with probability $\mathbb{P}(X = 2j + 1)$.

Then, $E_i$ is simply the sum over all $\delta_1(j)$, for all $j$. Since the block size is $k_1$, possible numbers of errors left are $2j = 0, 2 \ldots k_1$. Therefore, we have,

$$E_i = \sum_{j=0}^{\frac{k_i}{2}} ((2j)(\delta_1(j))\,.$$

□

**Theorem 4.3.** *We can approximate $\gamma_i$ for $i > 1$ with the lower-bound,*

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

*Proof.* Notice that $\gamma_i$ is the probability of correcting at least 2 errors in $B_x^1$ for $i > 1$, given that there are errors in $B_x^1$. This is the complement of the probability of not correcting any errors in $B_x^1$, given that there are errors in $B_x^i$. In other words,

$$\gamma_i = 1 - \mathbb{P}\left(\text{not correcting any errors in } B_x^1 \mid \text{ there are errors in } B_x^1\right).$$

Then, we have that,

$$\begin{aligned}
&\mathbb{P}\left(\text{not correcting any errors in } B_x^1 \mid \text{ there are errors in } B_x^1\right) \\
&\leq \mathbb{P}\left(\exists \text{ errors } e_1, e_2 \in B_x^1,\, e_1 \in B_x^i, e_2 \in B_y^i \text{ and } B_x^i, B_y^i \text{ have even parity}\right) \\
&\leq \left(\mathbb{P}\left(\exists \text{ at least 1 more error in } B_x^i \mid \text{there is an error in some } B_x^i\right)\right)^2 \\
&= \left(1 - \mathbb{P}\left(\text{there is only 1 error in some } B_x^i\right)\right)^2 \\
&= \left(1 - \mathbb{P}\left(\text{all } \frac{nE_{i-1}}{k_1} - 1 \text{ errors are outside } B_x^i\right)\right)^2. \qquad (4.4)
\end{aligned}$$

We then note that the fraction of the total key made up by a single block in the $i^{th}$ pass is $\frac{k_i}{n}$. Therefore,

$$\left(1-\mathbb{P}\left(\text{all } \frac{nE_{i-1}}{k_1}-1 \text{ errors are outside } B_x^i\right)\right)^2 \tag{4.5}$$

$$\approx \left(1-\left(1-\frac{k_i}{n}\right)^{\frac{nE_{i-1}}{k_1}}\right)^2. \tag{4.6}$$

A corollary of the prime number theorem,

$$e^a = \lim_{b\to\infty}(1-\frac{a}{b})^b$$

implies that,

$$e^{-ab} = \lim_{b\to\infty}(1-a)^b.$$

Thus, we have that,

$$\gamma_i \geq 1-\left(1-\left(1-\frac{k_i}{n}\right)^{\frac{nE_{i-1}}{k_1}}\right)^2 \approx 1-\left(1-e^{-\frac{k_iE_{i-1}}{k_1}}\right)^2. \tag{4.7}$$

□

**Theorem 4.8.** *For some pre-defined $n$ and $p$, if we assume that*

$$E_1 \leq -\frac{\ln\frac{1}{2}}{2}, \tag{4.9}$$

$$\sum_{l=j+1}^{\frac{k_i}{2}} \delta_i(l) \leq \frac{1}{4}\delta_i(j), \textit{for all } j,\ i,\ k_i, \tag{4.10}$$

*and,*

$$E_i \leq \frac{E_{i-1}}{2}, \tag{4.11}$$

*are true for all $i$ and $j$, then we have,*

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

*Proof.* Let $Y_i$ be the random variable denoting the number of errors in some $B_x^1$ after the $i^{th}$ pass. Then, notice that,

$$\begin{aligned}
\delta_i(j) &= \mathbb{P}\left((Y_{i-1}=2j)\cap(Y_i=2j)\right)+\mathbb{P}\left((Y_{i-1}-Y_i>0)\cap(Y_i=2j)\right)\\
&= \delta_{i-1}(l)\left(1-\gamma_i\right)+\mathbb{P}\left((Y_{i-1}-Y_i>0)\cap(Y_i=2j)\right)\\
&\leq \delta_{i-1}(l)\left(1-\gamma_i\right)+\mathbb{P}\left(Y_{i-1}>2j\right)\\
&= \delta_{i-1}(l)\left(1-\gamma_i\right)+\sum_{l=i+1}^{\frac{k_i}{2}}\delta_{i-1}(l)
\end{aligned}$$

Given that $k_i = 2k_{i-1}$ and (4.10), we have

$$\begin{aligned}\delta_i(j) &\leq \delta_{i-1}(l)\left(1-e^{-\frac{k_i E_{i-1}}{k_1}}\right)^2+\frac{1}{4}\delta_{i-1}(l) \\ &= \delta_{i-1}(l)\left(\left(1-e^{-\frac{k_1 E_{i-1}}{2^{i-1}k_1}}\right)^2+\frac{1}{4}\right) \\ &= \delta_{i-1}(l)\left(\left(1-e^{-\frac{E_{i-1}}{2^{i-1}}}\right)^2+\frac{1}{4}\right)\end{aligned} \tag{4.12}$$

Now, as assumed in (4.11),

$$\begin{aligned} & E_{i-1} \leq 2E_i \\ \implies & E_1 \leq 2^i E_i\end{aligned}$$

By this and (4.9) we have,

$$\begin{aligned}\delta_i(j) &\leq \delta_{i-1}(j)\left(1-e^{-\frac{E_{i-1}}{2^{i-1}}}\right)^2+\frac{1}{4}\delta_{i-1}(j) \\ &\leq \delta_{i-1}(j)\left(\left(1-e^{-\frac{2^{i-1}E_1}{2^{i-1}}}\right)^2+\frac{1}{4}\right) && \text{by (4.11)} \\ &\leq \delta_{i-1}(j)\left(\left(1-e^{E_1}\right)^2+\frac{1}{4}\right) \\ &\leq \delta_{i-1}(j)\left(\left(1-e^{-\frac{\ln\frac{1}{2}}{2}}\right)^2+\frac{1}{4}\right) && \text{by (4.9)} \\ &= \frac{\delta_{i-1}(j)}{2} &&,\end{aligned}$$

and we are done.

□

After having clarified Brassard's proof for a mathematically inclined audience, in the following two subsections, we hope to improve its rigor. We will present a list of possible reasons for Brassard's approximation not being a lower-bound for some values of $n$ and $p$, as has been shown in [6]. This shows that Brassard's bound is, in fact, not a bound but a general approximation.

4.2. **Inconsistencies and Inaccuracies.** In writing up the above proof, we had to make two major alterations from the proof in [3]. Firstly, there is what seems to be a rather untimely typographic error. In Brassard's proof, the bound (4.10) in Theorem 4.8 is assumed only for $i = 1$. This means that what we stated above as

$$\sum_{l=j+1}^{\frac{k_i}{2}} \delta_i(l) \leq \frac{1}{4}\delta_i(j),$$

was stated by Brassard as,

$$\sum_{l=j+1}^{\frac{k_i}{2}} \delta_1(l) \leq \frac{1}{4}\delta_1(j).$$

Brassard's version of this assumption renders the substitution at (4.12) to be incorrect for $i \neq 1$. Also, because the equation does not involve a strict equality sign, we cannot create an induction to derive the assumption we require from the one Brassard proposes.

Secondly, we present (4.11), namely that $E_i \leq \frac{E_{i-1}}{2}$ as an assumption in the statement of (4.8). In [3], the equation is cited as a fact, and used without justification. However, notice that $E_i \leq \frac{E_{i-1}}{2}$ is not universally true for all $i$, $p$ and $n$. For example, consider a block where every bit is an error, meaning that $p = 1.0$. Then, no errors would get corrected in any pass and $E_i = E_{i-1}$ for all passes. In this case, the equation $E_i \leq \frac{E_{i-1}}{2}$ does not hold. We cannot determine under what circumstances Brassard quoted this equation, but it is clearly not true in the general case.

There are a number of other inaccuracies in the proof as well. These do not require us to restructure or change the proof, as the above two did. We present them as Brassard did. For instance, the approximation at (4.7), takes the limit as $n$ goes to infinity. For smaller values of $n$, this approximation is inaccurate, as was shown in the work of [6]. Hence, Brassard's calculations cannot reliably be considered a lower-bound.

Also, in (4.6) Brassard makes an approximation. He approximates the probability of all $\frac{nE_{i-1}}{k_1} - 1$ errors outside $B_x^i$ to be,

$$\left(1 - \left(1 - \frac{k_i}{n}\right)^{\frac{nE_{i-1}}{k_1}}\right)^2,$$

instead of,

$$\left(1 - \left(1 - \frac{k_i}{n}\right)^{\frac{nE_{i-1}}{k_1}-1}\right)^2.$$

We are unsure how much of an effect this has on the value of $\gamma_i$, however, in [3], Brassard does not even present this as an approximation.

## 5. Possible Extensions

We propose the following as extensions to this project:

- *Extending the above calculations and bounds to $L_i$*: This paper reviews past work on $P_i(0)$, which extends naturally to a discussion on $L_i$.
- *Metric for comparison*: Suppose similar research is done for $L_i$, one can study an appropriate metric to discuss the trade-off between $L_i$ and $P_i(0)$ to optimize the protocol for various applications.
- *Choice of $k_1$*: Intuitively, it follows that $k_1 \approx \frac{1}{p}$ would optimize the number of blocks, at pass 1, with exactly one error. However, [4] states that empirical data suggests the use of $k_1 \approx \frac{0.73}{p}$. We could study the possible mathematical basis for this and the impact of this choice on $P_i(0)$ and $L_i$.

- *On $k_i = 2k_{i-1}$*: While convenient, [1] suggests that this is not the most efficient formula. One could seek a formula that is more efficient to determine $k_i$.
- *Question 2.5*: More research or simulation can be done to formalize or refute the claims sketched in Section 2.4. Specifically, it would be interesting to find out the circumstances under which each of the two `CASCADE` variants is more efficient.
- *Leaked bit calculations*: Leaked bit calculations, such as those proposed in [5] and [3], do not seem to acknowledge the point in (1.5). Namely, because the protocol is completed before leaked bits are removed, the same bit of information could have been leaked (as defined in (1.4)) multiple times. Therefore, when we go through the passes to remove some leaked bits, $\mathbb{E}(\sum L_i)$ is an expected upper-bound on the number of bits that need to be removed. Hence, future work on computing $\mathbb{E}\left(\sum L_i\right)$ can incorporate this.
- *Protocol modification*: The removal of leaked bits between passes instead of at the end seems like it could minimize computation that needs to be done on bits that will eventually be discarded. The key could additionally be padded with known bits to make it up to be a power of 2, or replaced with more bits from the quantum channel. [4] also cites literature that, much like the initial protocol proposed in [2], some protocols do not check through all the blocks in all passes. Further research can be done to evaluate the effectiveness of such changes.
- In discussing the inconsistencies in Brassard's proof, we focused on the reasons that the proof may not be a lower-bound, and may not have comprehensively discussed why it is a poor lower-bound, for values of $n$, $p$ and $i$ where it is a lower-bound. In particular, in the list of "inaccuracies" highlighted above, the reader is welcome to run simulations or theoretical research to determine if these substantially alter Brassard's conclusions.
- In Appendix A, we test the assumptions made by Brassard in [3]. However, we do not test these for the assumptions as modified in Section 4.2. This is described at length in Appendix A, but future work can attempt to explain why Brassard makes these assumptions, via proof or simulation.

**Acknowledgments.** This research was originally done at DSO National Laboratories, together with Khoongming Khoo and Sean Seet. It is a pleasure to thank my mentor, Rediet Abebe, for her guidance and cheer throughout this REU process. We would also like to thank Antonio (Tuca) Auffinger and Yan Zhang for taking time to help me analyze the protocol and Brassard's Proof. Finally, I would like to thank the organizers of the REU, Professor J. Peter May, Benjamin Fehrman and Jessica Lin.

## References

[1] Patrick Bellot and Minh-Dung Dang. BB84 Implementation and Computer Reality. RIVF '09. 2009.
[2] Charles H. Bennett, François Besette, Gilles Brassard, Louis Salvail and John Smolin. Experimental Quantum Cryptography. Springer-Verlag. 1992.
[3] Gilles Brassard and Louis Savail. Secret-Key Reconciliation by Public Discussion. Springer-Verlag. 1994.
[4] Timothy I. Calver. An Empirical Analysis of `CASCADE` Secret Key Reconciliation Protocol for Quantum Key Distribution ACM. 2011.

[5] Sean Seet, Ruth Ii-Yung Ng and Khoongming Khoo. An Accurate Analysis of the BINARY Information Reconciliation Protocol by Generating Functions. 2013.qcrypt.net/contributions/Seet-abstract.pdf. 2013.
[6] Koichi Yamazaki, Ranjith Nair and Horace P. Yuen. Problems of the CASCADE Protocol and Renyi Entropy Reduction in Classical and Quantum Key Generation. http://arxiv-web3.library.cornell.edu/pdf/quant-ph/0703012.pdf. 2008.

## Appendix A. Consistency of Brassard's Assumptions with Code-Runs

To add yet more rigor to Brassard's calculations, we wrote computer code to test the two assumptions in [3], namely (4.9) and (4.10). To do this, we test a range of values of $n$ and $p$ as would commonly be used in commercial Quantum Key Distribution systems. Since we lack the computational capacity and the mathematical machinery to compute the actual probability distribution at present, we test these assumptions against simulations of the CASCADE protocol. Finally, we corroborate [5] and [6] with more tests comparing $P_i(0)$ in simulation and in Brassard's bound for CASCADE.

Notice that we do not test (4.10) for $i \neq 1$, nor the additional assumption (4.11), both of which are required for a rigorous proof as discussed in Section 4.2. We do this because this paper is intended as a supplement to [3]. Therefore, we only test assumptions made in the original paper. Future work can expand on this.

A.1. **Testing $E_1 \leq -\frac{\ln \frac{1}{2}}{2}$.** Equation 4.9 states that,

$$E_1 \leq -\frac{\ln \frac{1}{2}}{2},$$

where we can recall from Theorem 4.2 that $E_1$ refers to the expected number of errors in some $B_x^1$ after pass 1 is completed. Notice that the value of $n$, the key-length, is independent of the value of $E_1$ and $\delta_1(j)$. Therefore, we test Brassard's hypothesis for varying values of $p$ and $k_1$. The results are summarized in Table A.1, with a 0 indicating that the hypothesis was not true and 1 indicating that it was.

| $p$ / $k_1$ | 0.01 | 0.03 | 0.05 | 0.07 | 0.09 | 0.11 |
|---|---|---|---|---|---|---|
| 8 | 1 | 1 | 1 | 1 | 1 | 0 |
| 16 | 1 | 1 | 0 | 0 | 0 | 0 |
| 32 | 1 | 0 | 0 | 0 | 0 | 0 |
| 64 | 1 | 0 | 0 | 0 | 0 | 0 |
| 128 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 1. Testing (4.9)

Therefore, we see that so long as the values of $k_1$ and $p$ are both small, Brassard's hypothesis in (4.9) is valid.

A.2. **Testing $\sum_{l=j+1}^{\frac{k_i}{2}} \delta_1(l) \leq \frac{1}{4}\delta_1(j)$.** Equation (4.10) states that given some $p$,

$$\sum_{l=j+1}^{\frac{k_i}{2}} \delta_i(l) \leq \frac{1}{4}\delta_i(j), \text{for all } j,\ i,\ k_i.$$

Notice that this equation is independent of $n$. In [3], this is assumed only for $i = 1$. We test this for a range of values of $j$, $k_1$, for some fixed $p$ and $i = 1$. This is done via the discrete equations in Theorem 4.2. The results are summarized in Tables 2, 3 and 4 below. As in the above section, an entry of 1 indicates that the hypothesis is true and an entry of 0 indicates that the hypothesis is not true. An entry of X indicates that the hypothesis is invalid with those specifications.

| $j$ / $k_1$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| 8 | 1 | 1 | 1 | 1 | 1 | X | X | X | X |
| 16 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 32 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 64 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 128 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 256 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

TABLE 2. Testing (4.9): $i = 1$, $p = 0.03$

| $j$ / $k_1$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| 8 | 1 | 1 | 1 | 1 | 1 | X | X | X | X |
| 16 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 32 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 64 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 128 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 256 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

TABLE 3. Testing (4.9): $i = 1$, $p = 0.05$

| $j$ / $k_1$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| 8 | 1 | 1 | 1 | 1 | 1 | X | X | X | X |
| 16 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 32 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 64 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 128 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 256 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

TABLE 4. Testing (4.9): $i = 1$, $p = 0.07$

In general, we can say that the assumption is true for large values of $j$, small values of $p$ and small values of $k_i$.

## Appendix B. Comparison of different Practical CASCADE Variants in (2)

A study of $P_i(0)$ and $L_i$ in simulation are summarized in the tables below for the industry standard of CASCADE and my variants of CASCADE as presented in (2), with each simulation conducted across 10000 randomly generated error positions and permutations.

| Value | BINARY | Industry Standard CASCADE | Our Variant of CASCADE |
|---|---|---|---|
| $P_1(0)$ | 0.2150 | 0.2150 | 0.2150 |
| $P_2(0)$ | 0.7800 | 0.8782 | 0.8746 |
| $P_3(0)$ | 0.9340 | 0.9699 | 0.9694 |
| $P_4(0)$ | 0.9655 | 0.9854 | 0.9825 |
| $P_5(0)$ | 0.9672 | 0.9862 | 0.9858 |
| $\sum_{i=1}^{1} L_i$ | 35.683201 | 35.683201 | 35.683201 |
| $\sum_{i=1}^{2} L_i$ | 51.705002 | 51.846802 | 51.802299 |
| $\sum_{i=1}^{3} L_i$ | 57.716801 | 56.8433001 | 56.827801 |
| $\sum_{i=1}^{4} L_i$ | 60.164001 | 59.022999 | 58.978901 |
| $\sum_{i=1}^{5} L_i$ | 61.143799 | 60.020000 | 60.021301 |

Table 5. Simulation Results for: $k_1 = 16$, $n = 256$, $p = 0.03$

| Value | BINARY | Industry Standard CASCADE | Our Variant of CASCADE |
|---|---|---|---|
| $P_1(0)$ | 0.0000 | 0.0000 | 0.0000 |
| $P_2(0)$ | 0.2764 | 0.8453 | 0.8407 |
| $P_3(0)$ | 0.9340 | 0.9957 | 0.9959 |
| $P_4(0)$ | 0.9655 | 0.9996 | 0.9997 |
| $P_5(0)$ | 0.9672 | 0.9999 | 0.9999 |
| $\sum_{i=1}^{1} L_i$ | 288.975189 | 288.975189 | 288.975189 |
| $\sum_{i=1}^{2} L_i$ | 213.558395 | 444.574310 | 444.644592 |
| $\sum_{i=1}^{3} L_i$ | 57.716801 | 478.483795 | 478.4950879 |
| $\sum_{i=1}^{4} L_i$ | 60.164001 | 494.551514 | 494.545685 |
| $\sum_{i=1}^{5} L_i$ | 61.143799 | 502.522308 | 502.576904 |

Table 6. Simulation Results for: $k_1 = 16$, $n = 2048$, $p = 0.03$

As discussed in Section 2.2, we see that BINARY is clearly an inferior protocol to CASCADE, and as discussed in Section 2.4, the industry standard and our variant of CASCADE are of comparable efficiency.