Interactive Error Correcting Codes Over Binary Erasure Channels Resilient to $> \frac{1}{2}$ Adversarial Corruption

Meghal Gupta *
Microsoft Research

Yael Tauman Kalai†
Microsoft Research and MIT

Rachel Yun Zhang‡
MIT

November 7, 2021

Abstract

An error correcting code (ECC) allows a sender to send a message to a receiver such that even if a constant fraction of the communicated bits are corrupted, the receiver can still learn the message correctly. Due to their importance and fundamental nature, ECCs have been extensively studied, one of the main goals being to maximize the fraction of errors that the ECC is resilient to.

For adversarial erasure errors (over a binary channel) the maximal error resilience of an ECC is $\frac{1}{2}$ of the communicated bits. In this work, we break this $\frac{1}{2}$ barrier by introducing the notion of an *interactive error correcting code* (iECC) and constructing an iECC that is resilient to adversarial erasure of $\frac{3}{5}$ of the total communicated bits. We emphasize that the adversary can corrupt both the sending party and the receiving party, and that both parties' rounds contribute to the adversary's budget.

We also prove an impossibility (upper) bound of $\frac{2}{3}$ on the maximal resilience of any binary iECC to adversarial erasures. In the bit flip setting, we prove an impossibility bound of $\frac{2}{7}$.

*E-mail:meghal@mit.edu

†E-mail:yael@microsoft.com

‡E-mail:rachelyz@mit.edu. Part of this work was done while at Microsoft Research. She is supported by an Akamai Presidential Fellowship.

Contents

1 Introduction

Consider the following task: Alice wishes to communicate a message to Bob such that even if a constant fraction of the communicated bits are adversarially tampered with, Bob is still guaranteed to be able to determine her message. This task motivated the prolific study of *error correcting codes*, starting with the seminal works of [Sha48, Ham50]. An error correcting code encodes a message x into a longer codeword $\mathsf{ECC}(x)$, such that the Hamming distance between any two distinct codewords is a constant fraction of the length of the codewords. To communicate a message x, Alice sends Bob the corresponding codeword $\mathsf{ECC}(x)$, and the fact that the distance between any two codewords is large guarantees that an adversary must corrupt a large fraction of the communication in order for Bob to decode to the wrong codeword.

An important question in the study of error correcting codes is determining the maximal possible error resilience. Indeed, many works focus on precisely this question. In general, the error resilience parameter depends on the alphabet size: in this work we focus on the binary alphabet. Two main error models are considered in the literature: The bit-flip model, where the adversary can flip a constant fraction of the bits of the codeword, and the erasure model, where the adversary can erase a constant fraction of the bits of the codeword (where each erased bit is replaced by a special erasure symbol).[1] It is known that in the adversarial bit-flip model, any ECC can be resilient to at most $\frac{1}{4}$ corruptions, and in the adversarial erasure error model any ECC can be resilient to at most $\frac{1}{2}$ corruptions.

In this work, we investigate the following natural question:

Can we achieve better error resilience if we use interaction?

This question is what motivated the study of ECC*'s with feedback* [Ber64], where after every bit sent the sender receives some information about the bits the receiver has received so far. This feedback is usually noiseless and does not count towards the adversary's budget: error resilience is measured as a fraction of Alice's forward rounds. Although noiseless feedback is known to be useful in increasing (forward) error resilience, little is known about noisy feedback in the setting of adversarial corruptions.

In our paper, we show that adversarially noisy feedback can be used to achieve forward erasure resilience $> \frac{1}{2}$. In fact, we show something even stronger: that adversarially erasable feedback can be used to achieve *total* erasure resilience $> \frac{1}{2}$ — for forward and feedback rounds, combined! We define the model of *interactive error correcting codes* (iECC), where the adversary is given a corruption budget which is an α-fraction of the total communication (for some $\alpha > 0$). She can spend it arbitrarily (e.g., all on forward communication, or on some combination of forward and feedback communication).[2] Quite surprisingly, we show that this model allows us to boost the erasure resilience of the protocol beyond $\frac{1}{2}$. In particular, we show that there is an iECC for which Bob correctly learns Alice's message x *even if* $\frac{3}{5}$ *of the total communication is erased!*

We view iECC's as the most general and natural solution to the original task of Alice communicating a message x to Bob under adversarial erasure. Note that a standard (non-interactive) ECC

[1] Another common model that was considered is the insertion/deletion model, which we do not address in this work.

[2] This is even more general than past considerations of noisy feedback [WQC17], where the adversary is given a separate budget to corrupt forward and feedback messages. We remark that all that was previously known in this setting is that the forward communication can be resilient to $\frac{1}{2}$ erasures — which is already achievable with standard (non-interactive) error correcting codes.

is an iECC in which Alice speaks in every round. Our result essentially shows that Bob talking occasionally *instead of Alice* actually improves the erasure resilience. It is not obvious that this should be the case! In particular, Bob can only send feedback since he has no input of his own, while Alice can actually send new information. Thus Bob's messages seem less valuable than Alice's. How can the erasure resilience be improved when messages containing information about x have been replaced by messages without? Nevertheless, we show that these feedback messages play a vital role in increasing the erasure resilience past $\frac{1}{2}$.

As mentioned above, we demonstrate that iECC's *are more powerful* than traditional ECC's by constructing an iECC that provides better error resilience for adversarial erasure corruptions. But there may be many other potential benefits they may have over traditional ECC's. Can iECC's achieve better error rate in other error models (such as the bit-flip model)? Do iECC's have any practical advantages? We leave these questions to investigation in future work.

1.1 Our Results.

Our main result is an iECC that achieves an erasure resilience of $\frac{3}{5}$ over the binary erasure channel.

Theorem 1.1. *For any $\epsilon > 0$, there exists an* iECC *over the binary erasure channel resilient to $\frac{3}{5} - \epsilon$ erasures, such that the communication complexity for inputs of size n is $O_\epsilon(n^2)$.*

We also prove the following impossibility result.

Theorem 1.2. *There does not exist an* iECC *over the binary erasure channel resilient to $\frac{2}{3}$ erasures.*

We also consider the problem of constructing an iECC over the binary *bit flip* channel. In this setting, instead of erasing bits, an adversary may toggle bits of her choice. We do not know of any constructions of iECCs that achieve a higher error resilience to bit flips than the standard (non-interactive) error correcting code (which achieves a maximal error rate of $\frac{1}{4}$). We leave open the problem of constructing such an iECC. We prove the following negative result on the maximal possible error resilience to bit flips of any iECC scheme.

Theorem 1.3. *There does not exist an* iECC *over the binary bit flip channel resilient to $\frac{2}{7}$ adversarial bit flips.*

1.2 Related Work

We mention two areas of research most pertinent to our work. The first is *error correcting codes with feedback* and the second is *interactive coding*.

1.2.1 Error Correcting Codes with Feedback

The notion of an *error correcting code with feedback* was first introduced in the PhD thesis of Berlekamp [Ber64]. In an error correcting code with feedback, Alice wishes to communicate a message to Bob in an error resilient fashion. For every message she sends, Bob sends her feedback. Originally, this feedback was considered in the *noiseless* setting, meaning that none of Bob's messages are allowed to be corrupted. Furthermore, Bob's feedback is *not* counted towards the adversary's corruption budget. That is, error rate is calculated solely as a function of the number of messages Alice sends.

In the bit flip error model, [Ber68, Zig76, SW92, HKV15] showed that the maximal error resilience of an error correcting code with noiseless feedback is $\frac{1}{3}$. For larger alphabets, the maximal error resilience was studied in [ADL06]. In the erasure model, error correcting codes with feedback over any alphabet are known to be resilient to an arbitrarily close to 1 fraction of erasures.[3]

When the feedback is *noisy*, i.e. the feedback may be corrupted as well, much less is known. Several works such as [BY08a, BY08b] considered ECC's with noisy feedback over the binary symmetric channel (each communicated bit is independently flipped with some probability). In the case of adversarial corruptions, the only work we know is that of [WQC17], which places separate corruption budgets on the forward and feedback rounds. They construct a scheme that is resilient to $\frac{1}{2}$ of the forward communication and 1 of the feedback being erased.[4] We note that their scheme's forward erasure resilience is equal to that achievable by standard error correcting codes.

As far as we know, our notion of an interactive error correcting code, where the adversary is permitted to freely corrupt both Alice's bits and Bob's feedback up to a constant fraction of the total communication, has not been studied before.

1.2.2 Interactive Coding

Interaction has been considered in the context of error resilience starting with the seminal works of Schulman [Sch92, Sch93, Sch96] and continuing in a prolific sequence of followup works, including [BR11, Bra12, BK12, BN13, Hae14, BE14, GHK+16, GH17, EGH16, EKS20]. These works consider the task of taking an *interactive* protocol and making it error resilient. That is, they consider *two-way* communication, instead of one-way as we do.

More specifically, Alice has a private input x, Bob has a private input y, and they both want to compute $f(x, y)$. Given a protocol π_0 that achieves this, but is not necessarily resilient to any error, the goal is to construct a (fixed order, fixed length) protocol π that is resilient to a constant fraction of adversarial bit flip errors. The question of maximizing π's error resilience has been studied in [BR11, GH13], albeit in the context of large alphabets. [BR11] presented the first error-resilient protocol over the binary channel, achieving an error resilience of $\frac{1}{8}$. This was later improved to $\frac{5}{39}$ in [EKS20], and finally to $\frac{1}{6}$ in the work of [GZ21]. $\frac{1}{6}$ is known to be optimal.

This problem was also considered in the case of adversarial erasures instead of adversarial bit flip errors [FGOS15, EGH16, GH17, GZ21]. The work of [FGOS15] originally constructed such a scheme over a large alphabet with erasure resilience $\frac{1}{2}$, which translates to a resilience of $\frac{1}{4}$ over the binary erasure channel by encoding each original letter with a binary error correcting code. The work of [EGH16] improved the binary erasure resilience to $\frac{1}{3}$, and the recent work of [GZ21] settled this question, showing that the optimal erasure resilience is $\frac{1}{2}$, which is known to be optimal [FGOS15].

We note that there are many other related works, some which consider adaptive speaking order and adaptive length protocols, where the adaptivity is added in an effort to improve the error resilience. There are also works on interactive coding in the multi-party setting. We refer the reader to [Gel17] for details.

[3] While we don't know of an explicit reference for this, this can be seen using ideas from this paper or from [GZ21]. Essentially Alice can send her input bit by bit, only moving onto the next bit when she receives confirmation that Bob has received the last bit.

[4] Our scheme achieves this as well.

2 Technical Overview

In this overview, we focus our efforts on our constructions of an iECC which bypass the $\frac{1}{2}$ error resilience barrier. We start with presenting a simplified construction that achieves erasure resilience of $\frac{6}{11}$, and then present our improved (complicated) construction that achieves erasure resilience of $\frac{3}{5}$. For the reader simply interested in how to construct iECC's better than traditional ECC's, it is sufficient to understand the $\frac{6}{11}$ protocol. For the impossibility bounds given in Theorems 1.2 and 1.3, we refer the reader to Sections 6.1 and 6.2 respectively.

2.1 A Simpler 6/11 iECC

We begin by discussing a simpler scheme that achieves an error resilience over the binary erasure channel of $\frac{6}{11} - \epsilon$.[5] Our starting point is based on the list-decoding to unique-decoding paradigm of [GH13, EKS20].

1. Alice sends Bob $\mathsf{ECC}(x)$, where ECC is an error correcting code of distance $\frac{1}{2}$.

   Denote by M the length of the codeword $\mathsf{ECC}(x)$. As we show in Lemma 3.3, as long as less than $\frac{3}{4}$ of the bits are erased, there are at most two codewords that agree with the unerased bits of $\mathsf{ECC}(x)$. Furthermore, since the adversary can only erase and not flip bits, we have a perfect guarantee that one of these codewords is $\mathsf{ECC}(x)$.

2. Bob sends Alice an index i on which the two inputs differ, using an error correcting code.

3. Alice decodes Bob's message, and replies with the value of her input on index i (which is sufficient for Bob to deduce x). She can do this by simply sending 0^M or 1^M. Bob now only needs to hear any *one* of Alice's message to learn her input.

Individually, both steps 1 and 3 are resilient to $\frac{3}{4}$ erasures, which gives hope for ultimately constructing a protocol resilient to $> \frac{1}{2}$ erasures. Unfortunately, there are some glaring issues. First, the fact that each message is resilient to $> \frac{1}{2}$ erasures does not imply that the final protocol is resilient to $> \frac{1}{2}$ erasures, since the adversary can choose to divide his corruption budget arbitrarily, and in particular can corrupt much more of one message at the cost of corrupting less of another. Furthermore, there is no obvious way for Bob to communicate i to Alice in a way resilient to $> \frac{1}{2}$ erasures. If Bob sends $\mathsf{ECC}(i)$, an adversary can simply erase half of Bob's message, making this step only $\frac{1}{2}$ error resilient.

This latter problem is addressed as follows: instead of having Bob send $\mathsf{ECC}(i)$, we limit Bob's message space to consist of only four possible messages, which can have relative distance $\frac{2}{3}$ (e.g. $\bar{0} = (000)^*, \bar{1} = (011)^*, \bar{2} = (101)^*, \bar{3} = (110)^*$). At this point the reader should ask: *How can Bob communicate the index i to Alice while only sending one of four possible messages?* To do this, we must use interaction. In our scheme, Bob communicates i to Alice via an *incrementation procedure* consisting of many rounds of interaction in which Bob always sends one of two codewords $\bar{0}, \bar{1}$. The other two codewords will be used to communicate some additional information that we will specify later.

[5] For simplicity, we omit ϵ's for the rest of this overview; all fractions r should be understood to be $r \pm O(\epsilon)$.

The incrementation procedure. In our incrementation procedure, Alice keeps track of a counter cnt initially set to 0 indicating her guess for i. In each round, she sends x and cnt, jointly encoded with a distance-$\frac{1}{2}$ error correcting code, to Bob. Bob's goal is to increment cnt to i by sending just two codewords $\bar{0}, \bar{1}$.

As a first attempt, one could consider a scheme in which Alice increments cnt every time she hears $\bar{1}$ from Bob, and stops incrementing when she hears $\bar{0}$ from Bob. However, there's a clear problem: what should Bob send if he doesn't hear Alice? He doesn't know if she has incremented enough yet, in which case he should send $\bar{0}$, or if she should increment again, in which case he should send $\bar{1}$. If he sends $\bar{1}$ every time he isn't sure, Alice might not know if Bob has heard her last message and wants her to keep incrementing or not, so she might increment past i. If he sends $\bar{0}$, the adversary could employ the following attack: she erases the $\bar{1}$ but not the following $\bar{0}$ from Bob, so that she only erases $\frac{1}{2}$ of Bob's messages (recall we need this to be $> \frac{1}{2}$), while keeping Alice from incrementing at all.

Instead, we use the following procedure: Alice increments only when she detects a *change* in Bob's message from $\bar{0}$ to $\bar{1}$ or vice versa. This change in Bob's message signals to her that Bob has heard her latest value of cnt and wants her to increment again; otherwise, he may not yet know if she's incremented or not. Meanwhile, Bob sends the same message $\bar{0}$ or $\bar{1}$ until he detects that Alice has incremented, before switching to sending the other codeword to ask Alice to increment again. The idea is that Alice will only increment again when Bob has acknowledged her previous incrementation and asked her to increment again, so that the two can never get out of sync. In particular, Alice cannot skip over the index i without Bob's permission.

Our protocol. Our protocol consists of many (say $\approx \frac{n}{\epsilon}$) *chunks*, where in each chunk Alice sends a message followed by Bob's reply. Our protocol is designed so that each such chunk will make *progress* towards Bob's unambiguously learning Alice's input, as long as the adversary did not invest more than $\frac{6}{11}$ error in that chunk. At a high level, in the first chunk with $< \frac{6}{11}$ erasures, Bob narrows down Alice's input to at most two options. In every future chunk with $< \frac{6}{11}$ erasures, either Alice gets closer to learning the index i on which the two options differ, or Bob fully determines x by ruling out one of the two values of x, e.g. by learning the value of $x[i]$ or by uniquely decoding Alice's message.

We choose the parameters so that in each chunk Alice sends a message of length M and Bob replies with a message of length $\frac{3}{8}M$. This choice implies that in a chunk with $< \frac{6}{11}$ erasures, it is guaranteed that either Bob hears $> \frac{1}{2}$ of Alice's message and thus can uniquely decode it, or he hears $> \frac{1}{4}$ of Alice's message *and* Alice hears $> \frac{1}{3}$ of Bob's message (this follows from the $8:3$ message length ratios of Alice and Bob). Therefore, in a chunk with $< \frac{6}{11}$ erasures, it is guaranteed that either Bob uniquely decodes Alice's message, or Bob narrows down Alice's message to two options and Alice uniquely decodes Bob's message (because Bob sends one of four codewords with relative distance $\frac{2}{3}$).

Let us describe the protocol. Alice keeps track of a counter cnt initially set to 0 indicating her guess for i. At the beginning of the protocol, Alice sends $\mathsf{ECC}(x, \mathsf{cnt})$ to Bob in every chunk. At some point there will be $< \frac{6}{11}$ erasures in a chunk, and so Bob list decodes Alice's message to at most two options, say $(x_0, \mathsf{cnt}_0 = 0)$ and $(x_1, \mathsf{cnt}_1 = 0)$. Since we are in the setting of erasures, one of the two decodings must be Alice's true state, and in particular must contain Alice's true input. Note that if instead Bob uniquely decodes Alice's message, he can unambiguously determine her input x. In general, the case where Bob uniquely decodes Alice's message allows him to trivially determine

x, so we do not mention it, and assume that in all chunks with $< \frac{6}{11}$ error, Bob list-decodes Alice's message to two options, and Alice uniquely decodes Bob's message.

At this point, Bob begins signaling to Alice to increment cnt. His goal is to tell Alice to increment cnt until $\mathsf{cnt} = i$. To do this, Alice increments cnt when she sees Bob's messages *change* from $\bar{0}$ to $\bar{1}$ or vice versa. Bob correspondingly waits until he next list-decodes Alice's message to two options, and sees that cnt has been incremented correctly before flipping his message. Because there are at least i chunks with $< \frac{6}{11}$ erasures, they will progress i times, and Alice will reach the index i.

Note the counters cnt_0 and cnt_1, obtained by list decoding Alice's message, may not be equal! In this case Bob will learn the correct x in a different way, as we explain later. For now assume that the two decoded messages are (x_0, cnt) and (x_1, cnt).

When Bob sees that Alice's counter has reached the index i, he begins sending a third codeword $\bar{2}$ to ask Alice for the value of her input at index i. Upon uniquely decoding $\bar{2}$, Alice knows that the index i has been reached and begins sending $x[i]$ for the rest of the protocol. As long as Bob eventually receives one bit of Alice's messages after this point, he can correctly deduce Alice's input x.

Recall, however, that at some point in the protocol the two messages that Bob decodes may have different counters $\mathsf{cnt}_0 \neq \mathsf{cnt}_1$. In order to learn x, it suffices for Bob to determine Alice's true value of cnt since x_b and cnt_b are paired up. If the first time the counters get out of sync one is at least 2 greater than the other, then Bob can conclude that one made an impossible increment and thus deduce x. Otherwise, the counters differ by 1, and Bob begins sending a fourth codeword $\bar{3}$, asking Alice to tell him the parity of her counter. This lets Bob deduce which of cnt_0 and cnt_1 was Alice's true counter, and then the corresponding x must be Alice's true input.

At the end of the protocol, Bob takes the last bit he ever received as the answer to his question (value of $x[i]$, corresponding to $\bar{2}$, or parity of cnt, corresponding to $\bar{3}$), and deduces Alice's input x. We refer the reader to Section 4 for a formal description of the protocol and its analysis.

2.2 Improving 6/11 to 3/5

In the above protocol, Bob uses the codewords $\bar{0}$ and $\bar{1}$ to increment Alice's counter. Then, when he wants to ask Alice for either the value of $x[\mathsf{cnt}]$ or the parity of cnt, he has to use two *new* codewords $\bar{2}$ or $\bar{3}$ to convey the appropriate question. By including these two extra codewords, the maximal possible distance between Bob's possible codewords decreases from 1 to $\frac{2}{3}$, which incurs a loss in error resilience. If we were able to somehow have Bob use only two codewords, the relative distance between Bob's codewords would increase to 1. Then, by letting Alice and Bob speak in a $4:1$ ratio, to prevent progress an adversary would have to either corrupt $\frac{3}{4}$ of Alice's message, or all of Bob's and $\frac{1}{2}$ of Alice's. This increases the erasure resilience to

$$\frac{\frac{3}{4} \cdot 4M}{4M + M} = \frac{M + \frac{1}{2} \cdot 4M}{4M + M} = \frac{3}{5}.$$

In order to get rid of the need for the extra two codewords $\bar{2}$ and $\bar{3}$, we need for the two codewords $\bar{0}$ and $\bar{1}$ to be able to take on *more than two meanings*. To do this, we combine our two codewords with *timing cues*, such that $\bar{0}$ and $\bar{1}$ mean different things depending on where they are heard in the protocol. More specifically, we divide the protocol into *blocks* of many messages, such that Alice interprets messages differently for the rest of the block depending on whether she first decoded a $\bar{0}$ or $\bar{1}$ within the block. The bits communicated in the rest of the block can now be combined with the first heard bit to take on more than two meanings.

There are two pieces of information that Bob needs to convey, which he previously used the extra codewords for. These are:

1. Telling Alice she is done incrementing cnt (and thus can switch to answering Bob's question for the rest of the protocol).

2. Telling Alice which question (value of $x[i]$ or parity of cnt) to answer.

First, we describe a new incrementation procedure, which ultimately allows Bob to tell Alice to stop incrementing the counter without introducing a new codeword. In this incrementation process, Alice always expects to hear $\bar{1}$ before $\bar{0}$ in every block, so she can reserve hearing $\bar{0}$ as the first message she hears in the block to mean that the incrementation is over. Then, we describe a second set of modifications to this protocol for Bob to specify his question without introducing a new codeword.

A new incrementation procedure. We partition the rounds into blocks, each consisting of *several* ($\approx \frac{1}{\epsilon}$) messages. The first message that Alice hears from Bob in each block indicates to her whether to increment her counter or terminate the incrementation stage: namely, Bob sends $\bar{1}$'s if he wants Alice to increment her counter and sends $\bar{0}$'s if he wants her to terminate the incrementation stage. However, if Bob just sends $\bar{1}$'s for the entire block when he wishes for Alice to increment (and $\bar{0}$'s when he wants her to terminate), we run into the same problem as we discussed earlier: Alice's messages may be erased, in which case Bob doesn't know whether Alice has incremented or not. To keep Alice and Bob in sync, Bob sends *confirmation* messages to tell Alice that he saw her previous incrementation and that she should increment again the next time she hears a $\bar{1}$.

To be precise, in each block, Bob attempts to increment Alice's cnt by exactly 1. When Bob wishes for Alice to increment her counter, he begins a block by sending $\bar{1}$'s to ask Alice to increment cnt, then when he sees that Alice has incremented cnt, he sends her $\bar{0}$'s for the rest of the block to *confirm* the incremented value of cnt. Alice only increments cnt again in the next block if her current value of cnt has already been confirmed.

To record Bob's confirmations, Alice has another variable cnfm, in addition to cnt, taking values in $\{\mathsf{true}, \mathsf{false}\}$. Each time Alice increments cnt she immediately sets $\mathsf{cnfm} \leftarrow \mathsf{false}$ until she gets a confirmation (a $\bar{0}$ after receiving at least one $\bar{1}$ within the same block) from Bob, at which point she sets $\mathsf{cnfm} \leftarrow \mathsf{true}$. Only when $\mathsf{cnfm} = \mathsf{true}$ does Alice increment cnt again when she receives a $\bar{1}$ from Bob.

Meanwhile, when Bob wishes to continue the incrementation stage he sends $\bar{1}$ in a block until he is *sure* that Alice heard him. Note that it is not always apparent from Alice's message whether she's heard him within this block or not, since if cnt is not confirmed she sends the same message $\mathsf{ECC}(x, \mathsf{cnt}, \mathsf{false})$ whether she heard a $\bar{1}$ this block or not. Therefore, to ensure that Bob receives feedback on whether Alice has heard a $\bar{1}$ this block, Alice adds yet another variable $\mathsf{rec} \in \{\mathsf{true}, \mathsf{false}\}$ indicating to Bob whether she's received a $\bar{1}$ from him this block.

In detail, our incrementation procedure is as follows:

- Every chunk, Alice sends her input x, along with a counter cnt, a boolean cnfm detailing if her current value of cnt is confirmed, and rec which is true if she has received a $\bar{1}$ so far this block, all jointly encoded with a standard error correcting code.

- Bob begins each block by sending $\bar{1}$'s.

- If the first message that Alice receives in a block is $\bar{1}$, she increments cnt and sets $\mathsf{cnfm} = \mathsf{false}$ if the previous value of cnt had been confirmed ($\mathsf{cnfm} = \mathsf{true}$); otherwise, she does nothing.
- If Bob learns that Alice has received a $\bar{1}$ this block, he attempts to confirm her value of cnt by sending $\bar{0}$ for the rest of the block.
- If Alice gets Bob's $\bar{0}$ after having received a $\bar{1}$ in the same block, she confirms her new value of counter by setting $\mathsf{cnfm} = \mathsf{true}$. She is now ready to increment again the next time she hears a $\bar{1}$.
- If the first message that Alice receives within a block is a $\bar{0}$, she moves on from this incrementation stage (which we later refer to as Stage 1) to a new stage, either Stage 2 or 3, which we will define and discuss later. Thus, when Bob wants Alice to terminate incrementation and move on (the equivalent of the old codewords $\bar{2}$ or $\bar{3}$) he sends $\bar{0}$'s the entire block.

We want to point to a technical issue that will complicate our protocol. As mentioned earlier, the adversary has the budget to erase $\frac{1}{2}$ of all of Alice's messages, and as a result confuse Bob between two Alices. What does Bob do when he receives a message from two possible Alices, one with $\mathsf{rec} = \mathsf{true}$ and the other with $\mathsf{rec} = \mathsf{false}$? Since the protocol must make progress in this case, we instruct Bob to send a confirmation (of the form $\bar{0}$) even if only one of these Alice's has $\mathsf{rec} = \mathsf{true}$.

The above choice can result in the following tricky situation: The adversary can erase Bob's first messages in a block, so that the "real" Alice does not receive any $\bar{1}$'s, and at the same time confuse Bob between two Alices, the real which has $\mathsf{rec} = \mathsf{false}$ and a fake which has $\mathsf{rec} = \mathsf{true}$, in which case, Bob will proceed to send confirmation of the form $\bar{0}$ for the rest of the block. The adversary will not erase these $\bar{0}$'s, and as a result the first message that the real Alice receives in the block is a $\bar{0}$, which will cause her to leave the incrementation stage of the protocol, even though cnt can be very far from i (and the counters of the two different Alice's equal). Recall that this situation could not occur in our $\frac{6}{11}$ protocol (described in Section 2.1), since we did not use the same codeword $\bar{0}$ to mean two different things!

To deal with this, Bob's goal will be more generally to guide the two Alices to send *different bits* by the end of the protocol, so that if he hears any such bit he can determine the real Alice. This is achieved via the question-asking paradigm, as we discuss next, but in certain edge cases, he uses different techniques as we discuss later.

Specifying the question. Recall that when the first message that Alice receives in a block is $\bar{0}$, she advances to Stage 2 or 3. Intuitively, the purpose of Stage 2 is for Alice to learn Bob's question (parity or value), and the purpose of Stage 3 is for Bob to learn Alice's answer. That is, when Alice is in Stage 3, she sends messages of the form b^{4M} where the bit b conveys her answer to Bob's question.

Alice learns Bob's question in Stage 2 similar to the way she learned the index i in Stage 1. Namely, she keeps yet another counter denoted by knt, whose purpose is similar to that of cnt in Stage 1. In the beginning of stage 2 knt is initialized to $\mathsf{knt} = 0$. She and Bob then participate in an incrementation procedure (similar to that in Stage 1), where the goal is to keep $\mathsf{knt} = 0$ if Bob wishes to learn the bit value $x[i]$, and increment it to 1 if he wishes to learn the parity of cnt.

We note that as opposed to the blocks which are of fixed length, and hence the parties always agree on when a block begins and when it ends, the length of each stage is not fixed and may depend

on Alice's input and the adversarial corruptions. As a result, Bob may not know which stage Alice is in. In an effort to remove this ambiguity, we double the signal of knt to also include which stage Alice is in. When Alice is in Stage 1 she sets $\mathsf{knt} = -1$, and when she is in Stages 1 and 2, she sends $\mathsf{ECC}(x, \mathsf{cnt}, \mathsf{cnfm}, \mathsf{rec}, \mathsf{knt})$ to Bob, so that if Bob receives a message with $\mathsf{knt} = -1$, he knows she is in Stage 1, and if $\mathsf{knt} \in \{0, 1\}$, he knows she's in Stage 2. When Alice is in Stage 3, her message is of the form b^{4M} for some $b \in \{0, 1\}$, which is also distinguishable.

Dealing with two different Alices. Unfortunately, the same problem of the adversary confusing Bob between two Alices continues to haunt us! Our tools from the $\frac{6}{11}$ protocol are sufficient only to deal with the case that both Alices are still in Stage 1, the incrementation stage. When this is not the case, e.g. the two Alices are in different stages, we must guarantee that they end up in Stage 3 with opposite bits.

To solve this, we first notice that in many cases Alice can skip Stage 2 altogether. For example, if the bit value $x[i]$ is equal to the parity of cnt, both of which are equal to 0, she can skip Stage 2 (since $x[i] = \mathsf{cnt} \bmod 2 = 0$ is the correct answer to both questions). Moreover, we slightly change the incrementation stage so that she can also skip Stage 2 in the case where the parity of cnt is 1. To this end, in the incrementation stage, Bob asks Alice to increment cnt until $\mathsf{cnt} = 2i$, in which case Alice knows he is interested in $x[\mathsf{cnt}/2]$ or in the parity of cnt. Thus, if cnt is odd there is no associated value question, so Alice knows to send 1, the parity of cnt, for the rest of the protocol.

With this change to the protocol, the only case where Alice actually needs to learn Bob's question is when her *parity* bit is 0 and her *value* bit is 1. In this case, Alice will advance from Stage 1 to Stage 2 by setting $\mathsf{knt} = 0$, and in all other cases Alice advances from Stage 1 directly to Stage 3 and sends her answer for the remainder of the protocol.

Now, lets go back to the question: *What does Bob do if the adversary confuses him between two Alices?* Bob deals with this differently, based on which stages the two Alices are in.

1. **Both Alices are in Stage 1.** We've already discussed this case: If either $\mathsf{cnt}_0 = \mathsf{cnt}_1 = 2i$ or $\mathsf{cnt}_0 \neq \mathsf{cnt}_1$, Bob sends $\bar{0}$ for the rest of the protocol. If $\mathsf{rec}_0 \neq \mathsf{rec}_1$, Bob behaves as if Alice heard his message (which unfortunately may cause the Alice with $\mathsf{rec} = \mathsf{false}$ to prematurely exit Stage 1).

2. **One Alice is in Stage 1 and the other is in Stage 2.** This is the difficult case! If Bob tries to increment both counters simultaneously and then advance both simultaneously, as he would if both were in Stage 1, he cannot necessarily coordinate them to send opposite bits for the rest of the protocol, since (as one of many issues) Stage 2 Alice may have left Stage 1 prematurely. We solve this problem by introducing a final layer of grouping, called the *megablock*.

   Our final protocol consists of several ($\approx \frac{1}{\epsilon}$) *megablocks*, each containing many ($\approx \frac{n}{\epsilon}$) blocks. At the beginning of each megablock, Alice resets the counter she is currently incrementing (either cnt or knt) to 0. If Bob ever sees two Alices, one of whom is in Stage 1 and the other who is in Stage 2, he waits until the start of the next megablock (meanwhile sending $\bar{1}$'s) and then sends $\bar{0}$ for the rest of the protocol. Recall that the Alice in Stage 2 must have $\mathsf{cnt} \bmod 2 = 0$ and $x[\mathsf{cnt}/2] = 1$, so when she receives a $\bar{0}$, she learns Bob's question to be *value* and sends 1 for the rest of the protocol. As for the Stage 1 Alice, by convention, we say that if Alice is in Stage 1 and receives a $\bar{0}$ first within a block while her value of cnt is 0, she advances directly to Stage 3 and sends 0 for the rest of the protocol. Thus, when Bob sends

$\bar{0}$ for the rest of the protocol starting at the beginning of a megablock, the Stage 1 Alice and the Stage 2 Alice both eventually advance to Stage 3 with opposite bits.

3. **Both Alices are in Stage 2.** We argue that this will never happen! (Unless it is easy for Bob to detect that one of the Alice's is fake, in which case he is done.) If Alice advances to Stage 2 prematurely (with $\mathsf{rec} = \mathsf{false}$), then the other Alice could not have advanced as well (since her $\mathsf{rec} = \mathsf{true}$). From this point on, Bob sends $\bar{1}$ until the start of the next megablock, and hence this second Alice must remain in Stage 1 the entire time. Then, when the new megablock starts, Bob sends $\bar{0}$ for the rest of the protocol, and this Alice can only either stay in Stage 1 or advance directly to Stage 3, as her $\mathsf{cnt} = 0$.

   In addition, Bob sends $\bar{0}$ for entire blocks only if it is the first time that $\mathsf{cnt}_0 = \mathsf{cnt}_1 = \mathsf{cnt}$ and $x_0[\mathsf{cnt}/2] \neq x_1[\mathsf{cnt}/2]$, or $\mathsf{cnt}_0 = \mathsf{cnt}_1 \pm 1$. In particular, it cannot be the case that $\mathsf{cnt}_0 = \mathsf{cnt}_1 = \mathsf{cnt} = 0 \pmod 2$ and $x_0[\mathsf{cnt}_0/2] = x_1[\mathsf{cnt}_1/2] = 1$, so it cannot be the case that both Alices advance to Stage 2.

4. **One of the Alice's is in Stage 3.** In this case, Bob only pays attention to the Alice that is *not* in Stage 3 (recall that the Stage 3 Alice ignores him anyway, and continues to send the same bit until the end of the protocol). His goal is to ensure that the other Alice arrives to Stage 3 with a bit different than that of the current Stage 3 Alice. This is easy to do since Bob has control over which bit Alice sends in Stage 3 (for example, if the other Alice is in Stage 1 he can ensure that she exits Stage 1 with $\mathsf{cnt} = 0$ if he wishes her to send 0, and $\mathsf{cnt} = 1$ if he wishes her to send 1; a similar strategy works for a Stage 2 Alice as well).

We refer the reader to Section 5 for the formal description of the protocol and its analysis.

2.3 Discussion

Both the iECC constructions presented in this paper involve the same high level idea: progress is made whenever Bob can narrow down Alice's message to two possibilities and Alice can decode Bob's message. Under this template, $\frac{3}{5}$ is in fact the optimal error resilience: an adversary can stall all progress by erasing $\frac{3}{4}$ of Alice's messages to confuse Bob between three states, or she can erase all of Bob's messages so that the iECC reduces to a non-interactive ECC and then erase $\frac{1}{2}$ of Alice's messages. Balancing the two attacks gives that an adversary can always succeed in confusing Bob with budget $\frac{3}{5}$.

We leave open the problem of whether our protocols can be modified to handle more than two worlds, so that progress is made whenever Bob list-decodes Alice's message to $k > 2$ possibilities and Alice uniquely decodes Bob's message. If this were the case, then an adversary could stall progress only if she erases $1 - \frac{1}{2^k}$ of Alice's message. As $k \to \infty$, the adversary must erase closer and closer to 1 of Alice's messages. In the limit, the erasure resilience approaches the bound given in Theorem 1.2.

3 Preliminaries and Definitions

Before we dive into the technical part of our paper, we present important preliminaries on classical error correcting codes, and define an iECC formally and what it means for one to be resilient to α-fraction of errors.

Notation. In this work, we use the following notations.

- The function $\Delta(x, y)$ represents the Hamming distance between x and y.
- The interval $[0, n]$ for $n \in \mathbb{Z}_{\geq 0}$ denotes the integers from 0 to n inclusive.
- The symbol $\perp$ in a message represents the erasure symbol that a party might receive in the erasure model.
- When we say Bob k-decodes a message, we mean that he list decodes it to exactly k possible messages Alice could have sent in the valid message space.

3.1 Classical Error Correcting Codes

Definition 3.1 (Error Correcting Code)**.** *An error correcting code (*ECC*) is a family of maps* $\mathsf{ECC} = \{\mathsf{ECC}_n : \{0,1\}^n \to \{0,1\}^{p(n)}\}_{n\in\mathbb{N}}$*. An* ECC *has* relative distance $\alpha > 0$ *if for all* $n \in \mathbb{N}$ *and any* $x \neq y \in \{0,1\}^n$*,*

$$\Delta\left(\mathsf{ECC}_n(x), \mathsf{ECC}_n(y)\right) \geq \alpha p.$$

where Δ is the Hamming distance. Binary error correcting codes with relative distance $\approx \frac{1}{2}$ are well known to exist with linear blowup in communication complexity.

Theorem 3.2 ([GS00])**.** *For all* $\epsilon > 0$*, there exists an explicit error correcting code* $\mathsf{ECC}_\epsilon = \{\mathsf{ECC}_{\epsilon,n} : \{0,1\}^n \to \{0,1\}^p\}_{n\in\mathbb{N}}$ *with relative distance* $\frac{1}{2} - \epsilon$ *and with* $p = p(n) = O_\epsilon(n)$*.*

A relative distance of $\frac{1}{2}$ is in fact optimal in the sense that as the number of codewords N approaches ∞, the maximal possible relative distance between N codewords approaches $\frac{1}{2}$. We remark, however, that for small values of N, the distance can be much larger: for $N = 2$, the relative distance between codewords can be as large as 1, e.g. the codewords 0^M and 1^M, and for $N = 4$, the relative distance can be as large as $\frac{2}{3}$, e.g. the codewords $(000)^M, (110)^M, (101)^M, (011)^M$. As mentioned in Section 2, our constructions leverage this fact that codes with higher relative distance exist for a small constant number of codewords.

We will also need the following important lemma about the number of shared bits between any three codewords in an error correcting code scheme that has distance $\frac{1}{2}$.

Lemma 3.3. *For any error correcting code* $\mathsf{ECC}_\epsilon = \{\mathsf{ECC}_{\epsilon,n} : \{0,1\}^n \to \{0,1\}^{p(n)}\}_{n\in\mathbb{N}}$ *with relative distance* $\frac{1}{2} - \epsilon$*, and any large enough* $n \in \mathbb{N}$*, any three codewords in* $\mathsf{ECC}_{\epsilon,n}$ *overlap on at most* $\left(\frac{1}{4} + \frac{3}{2}\epsilon\right) \cdot p$ *locations.*

Proof. Consider three codewords c_1, c_2, c_3. Suppose that all pairs are relative distance at least $\left(\frac{1}{2} - \epsilon\right)$. Let c_1 and c_2 share $f \leq \left(\frac{1}{2} + \epsilon\right) \cdot p$ bits, and all three codewords share $e \leq f$ bits. Then, note that

$$\begin{aligned} 2p \cdot \left(\frac{1}{2} - \epsilon\right) &\leq \Delta(c_1, c_3) + \Delta(c_3, c_2) \\ &\leq 2(f - e) + (p - f) \\ &= p + f - 2e \\ &\leq \left(\frac{3}{2} + \epsilon\right) \cdot p - 2e, \end{aligned}$$

which means that

$$e \leq \left(\frac{1}{4} + \frac{3}{2}\epsilon\right) \cdot p,$$

as claimed. □

Lemma 3.3 means that assuming that $< \frac{3}{4}$ of a codeword is erased, the resulting message is list-decodable to a set of size ≤ 2, at least in theory. The following theorem says that this list-decoding is polynomial time.

Theorem 3.4. *[Gur03, GS00] For all $\epsilon > 0$, there exists an explicit error correcting code $\mathsf{ECC}_\epsilon = \{\mathsf{ECC}_{\epsilon,n} : \{0,1\}^n \to \{0,1\}^p\}_{n\in\mathbb{N}}$ with relative distance $\frac{1}{2} - \epsilon$ and $p = p(n) = O_\epsilon(n)$, and a $\mathrm{poly}_\epsilon(n)$-time decoding algorithm $\mathsf{DEC}_\epsilon = \{\mathsf{DEC}_{\epsilon,n} : \{0,1\}^p \to \mathcal{P}(\{0,1\}^n)\}_{n\in\mathbb{N}}$, such that for any $n \in \mathbb{N}$, $x \in \{0,1\}^n$, and corruption σ consisting of fewer than $(\frac{3}{4} - \frac{3}{2}\epsilon) \cdot p$ erasures,*

$$|\mathsf{DEC}_{\epsilon,n}(\sigma \circ \mathsf{ECC}_{\epsilon,n}(x))| \leq 2, \qquad x \in \mathsf{DEC}_{\epsilon,n}(\sigma \circ \mathsf{ECC}_{\epsilon,n}(x)).$$

Furthermore, for all $n \in \mathbb{N}$, all codewords $\in \mathsf{ECC}_{\epsilon,n}$ are relative distance $\frac{1}{2} - \epsilon$ from the strings 0^p and 1^p, and from the strings $(000)^{p/3}, (011)^{p/3}, (101)^{p/3}, (110)^{p/3}$.[6]

3.2 Interactive Error Correcting Codes

We formally define our notion of an *interactive error correcting code (*iECC*)*. The two types of corruptions we will be interested in are erasures and bit flips. We first start by defining a non-adaptive interactive protocol.

Definition 3.5 (Non-Adaptive Interactive Protocol)**.** *A non-adaptive interactive protocol $\pi = \{\pi_n\}_{n\in\mathbb{N}}$ is an interactive protocol between Alice and Bob, where in each round a single party sends a single bit to the other party. The order of speaking, as well as the number of rounds in the protocol, is fixed beforehand. The number of rounds is denoted $|\pi|$.*

Definition 3.6 (Interactive Error Correcting Code)**.** *An interactive error correcting code (*iECC*) is a non-adaptive interactive protocol $\pi = \{\pi_n\}_{n\in\mathbb{N}}$, with the following syntax:*

- *At the beginning of the protocol, Alice receives as private input some $x \in \{0,1\}^n$.*
- *At the end of the protocol, Bob outputs some $\hat{x} \in \{0,1\}^n$.*

We say that π is α-resilient to adversarial bit flips (resp. erasures) if there exists $n_0 \in \mathbb{N}$ such that for all $n > n_0$ and $x \in \{0,1\}^n$, and for all online adversarial attacks consisting of flipping (resp. erasing) at most $\alpha \cdot |\pi|$ of the total communication, Bob outputs x at the end of the protocol with probability 1.

4 Interactive Error Correcting Codes Resilient to 6/11 Erasures

4.1 Protocol Outline

Recall that our protocol consists of many equal-length *chunks*, each consisting of M rounds from Alice to Bob followed by $\frac{3}{8}M$ rounds from Bob to Alice. We say that all the rounds spoken by a single party within a chunk is a *message*.

[6] This last property can be made to hold by taking an appropriate inner code.

We assume in the following description that Bob never uniquely decodes Alice's message, since then he trivially learns x. We also assume that whenever there are exactly two possible values of Alice's message compatible with what Bob received, that they are consistent with previous message pairs received by Bob, otherwise Bob can rule out one of the messages and successfully learn x.

1. Alice holds a counter cnt initially set to 0. Alice begins the protocol by sending $\mathsf{ECC}(x, \mathsf{cnt})$ to Bob in each chunk.

2. Bob begins the protocol by sending $\bar{0}$ if not otherwise specified. Let m be the (partially erased) message Bob receives from Alice. If Bob cannot list-decode m into at most two options, Bob ignores the message by simply sending the same message as he sent last. The first time that Bob list-decodes m into two possible options, he sets an index i on which they differ, and both possibilities must have $\mathsf{cnt} = 0$.

   Whenever he list-decodes into two options, he increments cnt or tells Alice to start sending him a single bit forever, as follows:

   - If the two values of Alice's counter cnt_0 and cnt_1 have both been incremented since the last time he 2-decoded and are still less than i, Bob switches his message to sending $\bar{1}$ if he had previously sent $\bar{0}$, and $\bar{0}$ if he had previously sent $\bar{1}$.
   - If the two values of Alice's counter cnt_0 and cnt_1 have both not been incremented since the last time Bob 2-decoded, Bob continues sending the same message he sent last.
   - If at any point the two values of Alice's counter have become off-by-1, i.e. one has incremented and the other hasn't, Bob sends the codeword $\bar{3}$ for the rest of the protocol to ask Alice for the parity of her counter cnt. Using the answer to this question, Bob can determine whether Alice's true input were x_0 or x_1.
   - Otherwise, $\mathsf{cnt}_0 = \mathsf{cnt}_1 = i$, where i is an index for which the two associated values of x differ. Bob then sends $\bar{2}$ for the rest of the protocol to ask Alice for her value of $x[\mathsf{cnt}]$, which will allow him to determine Alice's true input.

3. Whenever Alice unambiguously sees a *change* in Bob's message from a $\bar{0}$ to a $\bar{1}$ or vice versa, she increments her counter by 1. She does so until she unambiguously receives a $\bar{2}$ or $\bar{3}$, at which point she sends her current value of $x[\mathsf{cnt}]$ or cnt mod 2, respectively for the rest of the protocol.

The protocol is presented formally in Section 4.2.

4.2 Formal Protocol

In this section, we describe the $\frac{6}{11}$ erasure resilient protocol formally.

Protocol 1 : Interactive Binary One Way Protocol Resilient to $\frac{6}{11} - \frac{14}{11}\epsilon$ Erasures

Let n be the size of the message $x \in \{0,1\}^n$ that Alice wishes to convey to Bob. Let $\mathsf{ECC}(\cdot,\cdot) : \{0,1\}^n \times [0,n] \to \{0,1\}^M$ be the error correcting code of relative distance $\frac{1}{2} - \epsilon$ from Theorem 3.4, such that every codeword is also distance $\frac{1}{2} - \epsilon$ from each of 0^M and 1^M. Let $\bar{0}, \bar{1}, \bar{2}, \bar{3}$ denote length $\frac{3}{8}M$ binary strings that have relative distance $\frac{2}{3}$ from each other (specifi-

cally, $(000)^{M/8}, (011)^{M/8}, (101)^{M/8}, (110)^{M/8}$). Our protocol consists of $T = \lceil \frac{n+1}{\epsilon} \rceil$ chunks of Alice sending an M bit message followed by Bob sending a $\frac{3}{8}M$ bit message. Throughout the protocol, the parties will choose a response based on which case applies to their received message m; if multiple apply, they choose the first case on the list.

Alice

In addition to x, Alice has an internal state consisting of

- A counter cnt that at the beginning of the protocol is set to 0.
- An internal state mes, originally set to $\bar{0}$, representing Bob's most recent message that successfully got through to her.

She begins the protocol by sending $\mathsf{ECC}(x, \mathsf{cnt})$ as the first message. Before every future message, she takes note of the latest message ($\frac{3}{8}M$ rounds) from Bob as $m \in \{0, 1, \perp\}^{3M/8}$, and determines her response as follows. Note that if $< \frac{2}{3}$ of the symbols in m are erasures, there is only one value of $s \in \bar{0}, \bar{1}, \bar{2}, \bar{3}$ consistent with the message m.

Case 1: $\geq \frac{2}{3}$ *of the symbols in* m *are* $\perp$.
Alice sends the same message as in the previous chunk.

Case 2: m *uniquely decodes to* $s \in \{\bar{0}, \bar{1}\}$.
If $s \neq \mathsf{mes}$, Alice increments cnt by 1 and sets mes to s. She then sends $\mathsf{ECC}(x, \mathsf{cnt})$. (In other words, cnt is the number of times that Alice detected a flip in Bob's messages from $\bar{0}$ to $\bar{1}$ or vice versa.)
If $s = \mathsf{mes}$, Alice sends the same message as in the previous chunk.

Case 3: m *uniquely decodes to* $s = \bar{2}$.
Alice sends 1^M if $x[\mathsf{cnt}] = 1$ and 0^M if $x[\mathsf{cnt}] = 0$ in all subsequent chunks, ignoring any future instructions.

Case 4: m *uniquely decodes to* $s = \bar{3}$.
Alice sends $(\mathsf{cnt} \bmod 2)^M$ in all subsequent chunks, once again ignoring any future instructions.

Bob

Bob holds a variable $\hat{x}$, initially set to $\emptyset$, that will be updated with his final output either at the end of the protocol or once he has unambiguously learns Alice's value of x. Once $\hat{x}$ is set to a value ($\neq \emptyset$), it will not be updated again. That is, Bob ignores any future instructions to update it. At the end of the protocol, Bob outputs $\hat{x}$. If at any point in the protocol $\hat{x}$ has already been set, Bob may send Alice any arbitrary message, say $\bar{1}$.
Bob also keeps track of the following values:

- Two values $\hat{x}_0$ and $\hat{x}_1$, to be set when Bob 2-decodes Alice's message for the first time.
- A fixed index i where $\hat{x}_0$ and $\hat{x}_1$ differ, set as soon as $\hat{x}_0$ and $\hat{x}_1$ are known.
- mes, representing the last message ($\bar{0}$ or $\bar{1}$) he sent Alice; mes is originally set to $\bar{0}$.
- last, representing the last value Bob heard of Alice's counter, originally set to 0.
- A value $\mathsf{ques} \in \{2, 3\}$, indicating whether Bob wants to know the value of $x[i]$ or the parity of Alice's counter, respectively. This is only set when Bob transitions from Phase 1 to Phase 2.
- A bit $\mathsf{par} \in \{0, 1\}$, used in Phase 3 only if ques is set to 3, and is set at the same time as ques.

Each chunk, Bob's outgoing message is one of four codewords: $\bar{0}$, $\bar{1}$, $\bar{2}$, or $\bar{3}$. He begins the protocol in Phase 1 and at some point may transition to Phase 2, but can never go back to Phase 1. In Phase 1, his goal is to increment Alice's counter value and may only ever send $\bar{0}$ or $\bar{1}$. In Phase 2, his goal is to tell Alice to switch to sending only a single final bit, and in this phase he may only ever send either $\bar{2}$ or $\bar{3}$, but not both.

Phase 1: In Phase 1, Bob's message to Alice is always either $\bar{0}$ or $\bar{1}$. Let Alice's most recent message be $m \in \{0, 1, \perp\}^M$. Bob determines his next message depending on which of the following cases m falls under. Note that if $< \frac{3}{4} - \frac{3}{2}\epsilon$ of the symbols in m have been erased, there are at most two values of $s \in \{\mathsf{ECC}(x, \mathsf{cnt})\}_{x,\mathsf{cnt}}$ consistent with the message m.

Case 1: *$\geq \frac{3}{4} - \frac{3}{2}\epsilon$ of the symbols in m are $\perp$.*

Bob simply sends Alice mes again.

Case 2: *m is ≤ 2-decoded where at most one element is of the form $\mathsf{ECC}(x, \mathsf{cnt})$. There cannot be zero values of this form, since Alice only sends messages of the form $\mathsf{ECC}(x, \mathsf{cnt})$ while she has heard only $\bar{0}$'s and $\bar{1}$'s. Bob can therefore uniquely decode m to $\mathsf{ECC}(x, \mathsf{cnt})$, which must be Alice's true message. He then sets $\hat{x} \leftarrow x$.*

Case 3: *m is 2-decoded to two states $\{\mathsf{ECC}(x_0, \mathsf{cnt}_0), \mathsf{ECC}(x_1, \mathsf{cnt}_1)\}$ for the first time.*

In this case, the only value Bob has sent so far is $\bar{0}$. Therefore, if $\mathsf{cnt}_0 \neq 0$ or $\mathsf{cnt}_1 \neq 0$ (in which case exactly one is true as one must be Alice's real state), then Bob sets $\hat{x} \leftarrow x_b$ where b is such that $\mathsf{cnt}_b = 0$.

Otherwise, $\mathsf{cnt}_0 = \mathsf{cnt}_1 = 0$. For $b \in \{0, 1\}$, Bob sets $\hat{x}_b \leftarrow x_b$. He also sets i to be an index where $\hat{x}_0$ and $\hat{x}_1$ differ. He sets $\mathsf{mes} = \bar{1}$ and sends mes.

Case 4: *m is 2-decoded to two states $\{\mathsf{ECC}(x_0, \mathsf{cnt}_0), \mathsf{ECC}(x_1, \mathsf{cnt}_1)\}$, not for the first time.*

Bob switches the order of the two states if need be, so that at least one of $x_0 = \hat{x}_0$ or $x_1 = \hat{x}_1$.

- If for some $b \in \{0, 1\}$ it holds that $x_b \neq \hat{x}_b$ or $\mathsf{cnt}_b \notin \{\mathsf{last}, \mathsf{last} + 1\}$, then Bob sets $\hat{x} \leftarrow x_{1-b}$.
- If $\mathsf{cnt}_0 = \mathsf{cnt}_1 = \mathsf{last}$, he sends mes.
- If $\mathsf{cnt}_0 = \mathsf{cnt}_1 = \mathsf{last} + 1$, he sets $\mathsf{last} := \mathsf{cnt}_0 = \mathsf{cnt}_1$. If the new value last is equal to the index i, Bob sets $\mathsf{ques} = 2$ and moves to Phase 2. Otherwise, $\mathsf{last} < i$. Bob sets mes to be the logical not of itself, that is, if $\mathsf{mes} = \bar{0}$ the new value of mes is $\bar{1}$, and vice versa. He sends mes.
- If $\mathsf{cnt}_0 \neq \mathsf{cnt}_1$, then $|\mathsf{cnt}_0 - \mathsf{cnt}_1| = 1$. Bob sets $\mathsf{ques} = 3$ and $\mathsf{par} = (\mathsf{cnt}_1 \bmod 2)$, and moves to Phase 2.

If at the end of the protocol Bob is still in Phase 1, he sets $\hat{x}$ to a random value of x.

Phase 2: Bob always enters Phase 2 with a value of $\mathsf{ques} \in \{2, 3\}$ permanently set. He sends $\overline{\mathsf{ques}}$ each chunk for the rest of the protocol.

At the end of the protocol, let d denote Bob's most recently received bit from Alice. If $\mathsf{ques} = 2$, he sets $\hat{x} \leftarrow \hat{x}_b$, where $\hat{x}_b$ is such that $\hat{x}_b[i] = d$. If $\mathsf{ques} = 3$, he sets $\hat{x} \leftarrow \hat{x}_b$, where b is 1 if $d = \mathsf{par}$ and 0 otherwise.

4.3 Analysis

Theorem 4.1. *Protocol 1 is resilient to a $\frac{6}{11} - \frac{14}{11}\epsilon$ fraction of erasures. For an input of size n, the total communication is $O_\epsilon(n^2)$. Alice and Bob run in $\text{poly}_\epsilon(n)$ time.*

Proof. The number of communicated bits is $T \cdot \frac{11}{8}M = \lceil \frac{n+1}{\epsilon} \rceil \cdot O_\epsilon(n) = O_\epsilon(n^2)$. The runtime of Alice and Bob is governed by the time it takes to list-decode, which by Theorem 3.4 is $\text{poly}_\epsilon(n)$. We focus on proving error resilience.

Suppose that Bob outputs an incorrect value of x. We will show that the adversary must have erased at least $\frac{6}{11} - \frac{14}{11}\epsilon$ of the communication.

First, we claim that if Alice ever uniquely decodes Bob's message in a chunk R to be $\bar{2}$ or $\bar{3}$ and Bob hears at least one bit from Alice in a chunk after R, then Bob will output the correct value of Alice's input x. This is because the final bit will convey to him either the value of $x[i]$ or the parity of cnt, which allows him to distinguish between $\hat{x}_0$ and $\hat{x}_1$. Furthermore, any bit Alice sends after the R'th chunk must in fact be $x[i]$ if $\mathsf{ques} = 2$, or $\mathsf{cnt} \bmod 2$ if $\mathsf{ques} = 3$, since she had previously uniquely decoded Bob's message to be $\bar{2}$ or $\bar{3}$, so Bob must have actually sent $\bar{2}$ or $\bar{3}$ respectively.

Let R be the first chunk in which Alice uniquely decodes Bob's message to be $\bar{2}$ or $\bar{3}$, and if such a chunk does not exist then let $R = T$. The argument above implies that if Bob outputs the incorrect value of x, it must be the case that either none of Alice's messages after chunk R got through to Bob.

By the definition of our protocol, in the first R chunks, Alice sends $\mathsf{ECC}(x, \mathsf{cnt})$ for some $\mathsf{cnt} \in \{0, 1 \ldots, i\}$. Since we assumed that Bob outputs $\hat{x} \neq x$, it must be the case that none of these messages can be uniquely decoded, and in particular at least $\frac{1}{2} - \epsilon$ of each of Alice's messages must be erased, otherwise Bob uniquely decodes Alice's message and sets $\hat{x}$ correctly. Let S be the number of these R chunks in which at least $\frac{3}{4} - \frac{3}{2}\epsilon$ of Alice's messages are corrupted. In the other $R - S$ chunks, between $\frac{1}{2} - \epsilon$ and $\frac{3}{4} - \frac{3}{2}\epsilon$ of Alice's messages are corrupted. We next argue that in at most $i + 1$ of these $R - S$ chunks, Bob's messages to Alice have a unique decoding. This is the case since whenever Alice uniquely decodes Bob's message, she increments her counter (or has just received a $\bar{2}$ or $\bar{3}$ and knows to begin sending a single bit for the rest of the protocol), and this change is heard by Bob since he can ≤ 2-decode Alice's message in each of these $R - S$ chunks. As we have established, Alice's counter never exceeds i, which implies that in at most $i + 1 \leq n + 1$ of these $R - S$ chunks, Bob's messages to Alice have a unique decoding. In the other $\geq R - S - n - 1$ chunks, Bob's message to Alice is at least $\frac{2}{3}$ corrupted. This gives a total corruption rate of at least

$$
\begin{aligned}
&\frac{(T-R)\cdot M + S\cdot(\frac{3}{4}-\frac{3}{2}\epsilon)M + (R-S)\cdot(\frac{1}{2}-\epsilon)\cdot M + (R-S-n-1)\cdot\frac{2}{3}\cdot\frac{3}{8}M}{T\cdot\frac{11}{8}M} \\
=&\frac{T - \frac{\epsilon}{2}S - (\frac{1}{4}+\epsilon)R - \frac{1}{4}(n+1)}{\frac{11}{8}T} \\
\geq&\frac{(\frac{3}{4}-\frac{3}{2}\epsilon)T - \frac{1}{4}(n+1)}{\frac{11}{8}T} \\
\geq&\frac{6}{11} - \frac{14}{11}\epsilon,
\end{aligned}
$$

where in the last step we use that $T = \lceil \frac{n+1}{\epsilon} \rceil$. □

5 Interactive Error Correcting Codes Resilient to 3/5 Erasures

We recommend the reader understand the protocol achieving a $\frac{6}{11}$-resilience to erasures given in Section 4 before reading this section.

5.1 Protocol Overview

We refer the reader to the technical overview for a more comprehensive introduction to our protocol. In this section, we recall a couple changes made for our new protocol as compared to the $\frac{6}{11}$ protocol, then give an outline of the protocol.

First, we introduce two new layers of round grouping on top of the chunks (which we recall are two messages, one from Alice and one from Bob). The first is the *block*, which consists of $O_\epsilon(n)$ chunks. The second is the *megablock*, which consists of $O_\epsilon(1)$ blocks. Alice and Bob increment throughout a megablock at most once per block, as opposed to once per chunk as in the $\frac{6}{11}$. At the end of a megablock, Alice resets her counter.

Furthermore, Bob now attempts to increment cnt to the value $2i$ instead of i. Alice now understands the *value* question to be asking for the value of $x[\mathsf{cnt}/2]$, and the *parity* question to be asking for the parity of cnt.

Alice can be in one of three stages: Stage 1, in which she increments cnt; Stage 2, in which she increments a new counter knt to learn Bob's question; and Stage 3, in which she simply sends the same bit β for the rest of the protocol. Alice begins the protocol in Stage 1. At some point, she either advances directly to Stage 3 or first advances to Stage 2 before advancing further to Stage 3.

Meanwhile, Bob can be in one of three phases depending on the stages of the two Alices he decodes to. He begins in Phase 1, in which both Alices are in Stage 1. If at some point he decodes a Stage 1 Alice and a Stage 2 Alice, he transitions to Phase 2. Otherwise, if at some point he decodes a Stage 3 Alice, he transitions to Phase 3. Once he has entered either Phase 2 or 3, he stays there for the rest of the protocol.

The protocol. The final protocol is as follows. We assume that whenever Bob list-decodes Alice's message to two states that both states are consistent with previous pairs of states, that is, they are possible values of Alice's current state if she had been in the previous state and had seen some subset of Bob's message since the last 2-decoding. Otherwise, if Bob list-decodes to states such that only one is consistent with a past state, he learns x and doesn't need to go through with the rest of the protocol.

1. At the start of each megablock, Alice and Bob engage in an incrementation procedure in which Bob attempts to increment Alice's counter cnt to the value $2i$. At the start of each megablock, Alice resets cnt, and Bob, knowing this, restarts the incrementation.
2. Eventually, one of two things will happen.
    - Bob begins sending $\bar{0}$'s for the rest of the megablock to tell both Alices to advance from Stage 1 to Stage 2 or 3.
    - At some premature point, before the conditions for asking both Alices to advance have been reached, Bob sees that exactly one of the Alices has already advanced. This is possible if at some point Bob sees that one Alice has received a $\bar{1}$ while the other has not, since in that case he sends $\bar{0}$ for the rest of the block.

3. Alice advances to Stage 2 or possibly skips directly to Stage 3 if the first message she receives in a block is a $\bar{0}$. In Stage 2, Alice attempts to figure out whether Bob wants to know the answer to the *parity* or *value* question. In Stage 3, she sends a single bit β representing the answer to this question forever.

   In order to figure out which of the stages to advance to, she looks at $x[\mathsf{cnt}/2]$ and cnt mod 2. If they are both the same bit b, she can automatically proceed to Stage 3, with $\beta = b$. If cnt is odd, she knows Bob is asking the *parity* question, so she can still move to Stage 3. Only if $x[\mathsf{cnt}/2] = 1$ and $\mathsf{cnt} \bmod 2 = 0$ does she enter Stage 2.

   In Stage 2, starting with the next megablock after first advancing to Stage 2, Alice increments knt instead of cnt, resets knt (and not cnt) at the start of every megablock, and advances to Stage 3 when she hears $\bar{0}$ as the first unerased message of a block. At that point, if $\mathsf{knt} = 0$ she advances to Stage 3 with $\beta = x[\mathsf{cnt}/2] = 1$, and otherwise with $\beta = \mathsf{cnt} \bmod 2 = 0$.

4. When Bob sees that at least one Alice has advanced to Stage 2 or 3, he sends $\bar{1}$'s for the rest of the megablock, and changes his strategy in the next megablock. Note that both versions of Alice cannot be in Stage 2, since then they would both have $\mathsf{cnt} \bmod 2 = 0$ and $x[\mathsf{cnt}/2] = 1$, but Bob only allows both Alices to advance if the two Alices have different values of cnt mod 2 or $x[\mathsf{cnt}/2]$.

   Case 1: *One Alice is in Stage 1, and the other is in Stage 2.*

   It is the beginning of the megablock so the Alice in Stage 1 has $\mathsf{cnt} = 0$ and the Alice in Stage 2 has $\mathsf{knt} = 0$. Bob simply sends $\bar{0}$ for the rest of the protocol, so that when the Alices advance to Stage 3, they do so with opposite bits: the Alice in Stage 1 answers 0, and the Alice in Stage 2 answers $x[\mathsf{cnt}/2] = 1$. This is known as Phase 2.

   Case 2: *One Alice is in Stage 1 or 2, and the other Alice is in Stage 3.*

   The answer $\beta \in \{0, 1\}$ of the Stage 3 Alice is fixed, so Bob can focus on incrementing the remaining Alice's counter to a value for which her answer to Bob's question is $1 - \beta$. This is known as Phase 3.

5. At the end of the protocol, Bob should have coordinated both versions of Alice to send opposite bits forever; hearing any one bit from Alice now suffices to deduce x.

We provide the formal protocol in the next section.

5.2 Formal Protocol

Protocol 2 : Interactive Binary One Way Protocol Resilient to $\frac{3}{5} - \epsilon$ Erasures

Let n be the size of the message $x \in \{0,1\}^n$ that Alice wishes to convey to Bob. Let

$$\mathsf{ECC}(x, \mathsf{cnt}, \mathsf{cnfm}, \mathsf{rec}, \mathsf{knt}, \mathsf{stg2}) : \{0,1\}^n \times [0,n] \times \{\mathsf{true}, \mathsf{false}\} \times \{\mathsf{true}, \mathsf{false}\} \times \{-1, 0, 1\} \times \{\mathsf{true}, \mathsf{false}\} \to \{0,1\}^{4M}$$

be an error correcting code of relative distance $\frac{1}{2} - \epsilon$ such that each codeword is also relative distance $\geq \frac{1}{2} - \epsilon$ from each of $0^{4M}, 1^{4M}$, as given in Theorem 3.4. Also, let $\bar{0} = 0^M$ and $\bar{1} = 1^M$. Our protocol

consists of $A = \lceil \frac{1}{\epsilon} \rceil$ megablocks of $B = \lceil \frac{n}{\epsilon} \rceil$ blocks, each consisting of $C = \lceil \frac{1}{\epsilon} \rceil$ chunks of $5M$ rounds, made of Alice sending a $4M$ bit message followed by Bob sending a M bit message. Throughout the protocol, the parties will choose a response based on which case applies to their received message m; if multiple apply, they choose the first case on the list.

Alice

Alice's message will always be from the following set:

$$\{\mathsf{ECC}(x, \mathsf{cnt}, \mathsf{cnfm}, \mathsf{rec}, \mathsf{knt}, \mathsf{stg2})\}_{x,\mathsf{cnt},\mathsf{cnfm},\mathsf{rec},\mathsf{knt},\mathsf{stg2}} \cup \{0^{4M}, 1^{4M}\}$$

which has relative distance $\geq \frac{1}{2} - \epsilon$. She tracks certain values explicitly, namely the *stage* of the protocol she is in, and the following variables:

- A counter cnt taking values in $[0, n]$, initially set to 0.
- A bit $\mathsf{cnfm} \in \{\mathsf{true}, \mathsf{false}\}$ denoting whether she has received a 0 from Bob after sending this particular value of cnt and later knt (i.e. whether she has received *confirmation* that $\mathsf{cnt}/\mathsf{knt}$ was successfully sent to Bob) within this block. Initially cnfm is set to true.
- A bit $\mathsf{rec} \in \{\mathsf{true}, \mathsf{false}\}$ denoting whether she has *received* a 1 so far this block. rec is initially set to false.
- A second counter knt taking values in $\{-1, 0, 1\}$. It will be used in Stage 2 to determine which question Bob wishes answered. It is initially set to -1 when Alice is in Stage 1. When Alice advances to Stage 2, she sets $\mathsf{knt} \leftarrow 0$.
- A boolean $\mathsf{stg2}$ indicating whether she advanced to stage 2 within this megablock. It is initially set false. $\mathsf{stg2}$ is used to ensure Alice does not advance stages twice within the same megablock.

Alice acts differently according to which of three stages she is in. In Stage 1, she increments her counter cnt; in Stage 2, she increments knt to learn whether Bob wants her to answer the parity or value question; and in Stage 3 she sends only the single bit β equal to the answer to Bob's question for the rest of the protocol. Alice may go directly from Stage 1 to 3, or pass through Stage 2 in the middle.

Stage 1: At the beginning of each megablock, she resets all the above variables to their initial states.

At the beginning of each block, she resets $\mathsf{rec} = \mathsf{false}$. As her first message of each block, she sends $\mathsf{ECC}(x, \mathsf{cnt}, \mathsf{cnfm}, \mathsf{rec}, \mathsf{knt} = -1, \mathsf{stg2} = \mathsf{false})$. Let $m \in \{0, 1, \perp\}^M$ be the most recent message from Bob; for each remaining message in the block, she determines what to send Bob based on which case m falls under. Note that if m is not entirely erased, then Alice can uniquely decode m to one of $\bar{0}$ and $\bar{1}$.

Case 1: *All of the symbols in m are $\perp$.*

Alice sends the same message as the last chunk: $\mathsf{ECC}(x, \mathsf{cnt}, \mathsf{cnfm}, \mathsf{rec}, \mathsf{knt} = -1, \mathsf{stg2} = \mathsf{false})$.

Case 2: *m decodes uniquely to $\bar{1}$.*

Alice sets $\mathsf{rec} \leftarrow \mathsf{true}$. If $\mathsf{cnfm} = \mathsf{true}$, she increments cnt and sets $\mathsf{cnfm} \leftarrow \mathsf{false}$. She sends her updated value of $\mathsf{ECC}(x, \mathsf{cnt}, \mathsf{cnfm}, \mathsf{rec}, \mathsf{knt} = -1, \mathsf{stg2} = \mathsf{false})$.

Case 3: *m decodes uniquely to $\bar{0}$, and $\mathsf{rec} = \mathsf{true}$ (i.e. she's already seen a 1 this block).*

Alice sets $\mathsf{cnfm} \leftarrow \mathsf{true}$. She sends $\mathsf{ECC}(x, \mathsf{cnt}, \mathsf{cnfm}, \mathsf{rec}, \mathsf{knt} = -1, \mathsf{stg2} = \mathsf{false})$.

Case 4: *m uniquely decodes to $\bar{0}$, and* $\mathsf{rec} = \mathsf{false}$ *(i.e. she has seen no messages this block and the first message received is a $\bar{0}$).*

If cnt is odd, she advances to Stage 3 of the protocol with $\beta = 1$.

Otherwise, cnt is even, and if $\mathsf{cnt} = 0$ or $x[\mathsf{cnt}/2] = 0$, she advances to Stage 3 with $\beta = 0$.

Else, cnt is even and $x[\mathsf{cnt}/2] = 1$. She advances to Stage 2.

Stage 2: Note that $x[\mathsf{cnt}/2] = 1$ and $\mathsf{cnt} \bmod 2 = 0$ in order for Alice to enter this stage.

As soon as Alice enters Stage 2, she sets $\mathsf{knt} \leftarrow 0$ and $\mathsf{stg2} = \mathsf{true}$. For the rest of the megablock, she sends $\mathsf{ECC}(x, \mathsf{cnt}, \mathsf{cnfm}, \mathsf{rec}, \mathsf{knt} = 0, \mathsf{stg2} = \mathsf{true})$, ignoring Bob's messages and not updating any of her values.

At the start of each successive megablock, Alice resets $\mathsf{cnfm} \leftarrow \mathsf{true}$, $\mathsf{rec} \leftarrow \mathsf{false}$, $\mathsf{knt} \leftarrow 0$, and $\mathsf{stg2} \leftarrow \mathsf{false}$. We point out that she never alters cnt again. At the beginning of each block, she resets $\mathsf{rec} \leftarrow \mathsf{false}$. She begins the block by sending $\mathsf{ECC}(x, \mathsf{cnt}, \mathsf{cnfm}, \mathsf{rec}, \mathsf{knt}, \mathsf{stg2})$. Let $m \in \{0, 1, \perp\}^M$ be the most recent message from Bob; she determines what to send next based on the following cases. As before, if at least one symbol in m is not $\perp$, then Bob's original message can only be one of $\bar{0}, \bar{1}$.

Case 1: *All of the symbols in m are $\perp$.*

Alice sends $\mathsf{ECC}(x, \mathsf{cnt}, \mathsf{cnfm}, \mathsf{rec}, \mathsf{knt}, \mathsf{stg2} = \mathsf{false})$.

Case 2: *m uniquely decodes to $\bar{1}$*

Alice sets $\mathsf{rec} \leftarrow \mathsf{true}$. If $\mathsf{cnfm} = \mathsf{true}$, she increments knt and sets $\mathsf{cnfm} \leftarrow \mathsf{false}$. She sends her updated value of $\mathsf{ECC}(x, \mathsf{cnt}, \mathsf{cnfm}, \mathsf{rec}, \mathsf{knt}, \mathsf{stg2} = \mathsf{false})$.

Case 3: *m uniquely decodes to $\bar{0}$, and* $\mathsf{rec} = \mathsf{true}$ *(i.e. she's already seen a 1 this block).*

Alice sets $\mathsf{cnfm} \leftarrow \mathsf{true}$. She sends $\mathsf{ECC}(x, \mathsf{cnt}, \mathsf{cnfm}, \mathsf{rec}, \mathsf{knt}, \mathsf{stg2} = \mathsf{false})$.

Case 4: *m uniquely decodes to $\bar{0}$, and* $\mathsf{rec} = \mathsf{false}$ *(i.e. she has seen no messages this block and the first message received is a $\bar{0}$).*

If $\mathsf{knt} = 0$, she advances to Stage 3 of the protocol with $\beta = x[\mathsf{cnt}/2] = 1$.

Otherwise, if $\mathsf{knt} = 1$, she advances to Stage 3 of the protocol with $\beta = (\mathsf{cnt} \bmod 2) = 0$.

Stage 3: When Alice reaches this stage, she has noted a bit β. She sends this for the rest of the protocol; all her messages from here on out are irreversibly β^{4M}.

Bob

Bob holds a variable $\hat{x}$, initially set to $\emptyset$, representing his final output. It will either be set at the end of the protocol or sometime during the protocol if he unambiguously learns Alice's value of x. In either case, once set it is final, and at the end of the protocol, Bob outputs $\hat{x}$. We remark that once $\hat{x}$ is set, Bob may behave arbitrarily for the rest of the protocol: let us say that he simply sends $\bar{1}$ for each message thereafter and does no further updates to any of his internal variables.

Bob tracks which phase he is in explicitly, and he holds certain values to help him perform consistency checks of 2-decoded Alice's messages with previous pairs:

- $\hat{x}_0$ and $\hat{x}_1$ storing two possible values of x Alice holds.

- S_0 and S_1 storing all possible messages Alice could send next that are consistent with previous 2-decoded pairs of messages, assuming she has received some subset of messages from Bob since his last 2-decoding, corresponding to her holding the input $\hat{x}_0$ and $\hat{x}_1$ respectively.

The values $\hat{x}_0, \hat{x}_1, S_0, S_1$ are initialized and updated as follows. He initializes $\hat{x}_b, S_b$ immediately upon receiving a 2-decodable message from Alice. If he has not yet set $\hat{x}$, he updates S_b each time he 2-decodes Alice's message and also after each each message he sends to Alice.

Initializing $\hat{x}_b, S_b$: The first time in the entire protocol that Bob receives a message from Alice that has $< \frac{3}{4} - \frac{3}{2}\epsilon$ erasures, Bob has only sent $\bar{1}$ so far, and he sets $\{\hat{x}_b, S_b\}_{b\in\{0,1\}}$, as follows:

- If Bob 1-decodes Alice's message to $\mathsf{ECC}(x, \mathsf{cnt}, \mathsf{cnfm} = \mathsf{false}, \mathsf{rec}, \mathsf{knt} = -1, \mathsf{stg2} = \mathsf{false})$, where $(\mathsf{cnt}, \mathsf{rec}) \neq (0, \mathsf{true})$, or both messages are of the form $\mathsf{ECC}(x_b, \mathsf{cnt}_b, \mathsf{cnfm}_b = \mathsf{false}, \mathsf{rec}_b, \mathsf{knt}_b = -1, \mathsf{stg2}_b = \mathsf{false})$ and $x_0 = x_1 = x$, Bob sets $\hat{x} \leftarrow x$.
- Otherwise, if Bob 2-decodes Alice's message to

$$\left\{\begin{array}{l}\mathsf{ECC}(x_0, \mathsf{cnt}_0, \mathsf{cnfm}_0 = \mathsf{false}, \mathsf{rec}_0, \mathsf{knt}_0 = -1, \mathsf{stg2}_0 = \mathsf{false}),\\ \mathsf{ECC}(x_1, \mathsf{cnt}_1, \mathsf{cnfm}_1 = \mathsf{false}, \mathsf{rec}_1, \mathsf{knt}_1 = -1, \mathsf{stg2}_1 = \mathsf{false})\end{array}\right\}$$

  both satisfying $(\mathsf{cnt}_b, \mathsf{rec}_b) \neq (0, \mathsf{true})$ such that $x_0 \neq x_1$, Bob sets $\hat{x}_0 \leftarrow x_0$ and $\hat{x}_1 \leftarrow x_1$. He also sets $S_0 \leftarrow \{\mathsf{ECC}(x_0, \mathsf{cnt}_0, \mathsf{cnfm}_0, \mathsf{rec}_0, \mathsf{knt}_0, \mathsf{stg2}_0)\}$ and $S_1 \leftarrow \{\mathsf{ECC}(x_1, \mathsf{cnt}_1, \mathsf{cnfm}_1, \mathsf{rec}_1, \mathsf{knt}_1, \mathsf{stg2}_1)\}$.

Updating S_b: Bob updates the sets S_0, S_1 whenever he receives a message from Alice that is $< \frac{3}{4} - \frac{3}{2}\epsilon$ erased, and also *after* each message he sends.

- Whenever Bob receives a message m from Alice with fewer than $\frac{3}{4} - \frac{3}{2}\epsilon$ erasures, if he does not set $\hat{x}$ in response, he must have 2-decoded m to two messages $m_0 \in S_0, m_1 \in S_1$. He sets $S_0 = \{m_0\}$ and $S_1 = \{m_1\}$.
- After sending a message m_B, Bob updates S_b as follows. For each $m_A \in S_b$, he computes the ≤ 2 next possible messages that Alice would send if she had last sent m_A and now either hears or doesn't hear m_B. The new set S_b consists of all such possible new messages.

  We remark that the message m_A is sufficient to compute the possible new messages: if $m_A = \mathsf{ECC}(\hat{x}_b, \mathsf{cnt}_b, \mathsf{cnfm}_b, \mathsf{rec}_b, \mathsf{knt}_b, \mathsf{stg2}_b)$, then the variables $\hat{x}_b, \mathsf{cnt}_b, \mathsf{cnfm}_b, \mathsf{rec}_b, \mathsf{knt}_b, \mathsf{stg2}_b$ are sufficiently to simulate Alice's behavior whether she hears m_B or not; and if $m_A = \beta^{4M}$ for some $\beta \in \{0, 1\}$, then the only possible new message that Alice sends is the same message $m_A = \beta^{4M}$.

Each of Bob's messages throughout the protocol is either $\bar{0}$ or $\bar{1}$. Bob begins the protocol in Phase 1, and eventually transitions to either Phase 2 or Phase 3, where he stays for the rest of the protocol. Bob behaves differently depending on which phase he's in.

Phase 1: In Phase 1, Bob's goal is to increment Alice's counter so that it reaches $2i$, where i is an index for which $x_0[i] \neq x_1[i]$. If this goal is reached, i.e. $\mathsf{cnt}_0 = \mathsf{cnt}_1 = \mathsf{cnt}$ and $x_0[\mathsf{cnt}/2] \neq x_1[\mathsf{cnt}/2]$, or the counters become misaligned, i.e. $\mathsf{cnt}_0 = \mathsf{cnt}_1 \pm 1$, he switches from incrementing Alice's counter to sending her $\bar{0}$'s for the rest of the megablock. If he ever sees that one of the two Alices has advanced to Stage 2 or 3, he transitions to Phase 2 or Phase 3 depending on the stages of the two Alices he sees.

At the start of every megablock in which Bob is still in Phase 1 at the beginning, Bob sends $\bar{1}$. He determines the message he sends next based on the following cases for the message $m \in \{0, 1, \perp\}^{4M}$ he most recently received from Alice:

Case 1: *m has at least $\frac{3}{4} - \frac{3}{2}\epsilon$ $\perp$'s.*

Bob repeats his last message, unless it is the first message of a block, in which case he sends $\bar{1}$.

Case 2: *Bob ≤ 2-decodes m to $\{m_0, m_1\}$, and $S_b \cap \{m_0, m_1\} = \emptyset$ for some $b \in \{0,1\}$.*

Bob sets $\hat{x} \leftarrow \hat{x}_{1-b}$.

In all remaining cases, Bob 2-decodes m to two messages $m_0 \in S_0$ and $m_1 \in S_1$.

Case 3: *One of the worlds has already advanced, that is, for some $b \in \{0,1\}$, either $m_b = \mathsf{ECC}(\hat{x}_b, \mathsf{cnt}_b, \mathsf{cnfm}_b, \mathsf{rec}_b, \mathsf{knt}_b, \mathsf{stg2}_b)$ with $\mathsf{knt}_b \in \{0,1\}$, or $m_b \in \{0^{4M}, 1^{4M}\}$.*

Bob sends $\bar{1}$ for every message for the rest of the megablock, ignoring any instructions to send any other messages. He still updates S_0, S_1 and checks consistency of m with S_b (Case 2).

If either m_0 or m_1 is in $\{0^{4M}, 1^{4M}\}$, Bob transitions to Phase 3 at the beginning of the next megablock. Otherwise, if $m_b = \mathsf{ECC}(\hat{x}_b, \mathsf{cnt}_b, \mathsf{cnfm}_b, \mathsf{rec}_b, \mathsf{knt}_b \in \{0,1\}, \mathsf{stg2}_b)$ for some $b \in \{0,1\}$, he transitions to Phase 2 at the beginning of the next megablock.

For the rest of the cases, $m_0 = \mathsf{ECC}(\hat{x}_0, \mathsf{cnt}_0, \mathsf{cnfm}_0, \mathsf{rec}_0, \mathsf{knt}_0 = -1, \mathsf{stg2}_0 = \mathsf{false})$ and $m_1 = \mathsf{ECC}(\hat{x}_1, \mathsf{cnt}_1, \mathsf{cnfm}_1, \mathsf{rec}_1, \mathsf{knt}_1 = -1, \mathsf{stg2}_1 = \mathsf{false})$.

Case 4: $\mathsf{cnt}_0 = \mathsf{cnt}_1 = 2i$ *or* $\mathsf{cnt}_0 \neq \mathsf{cnt}_1$.

Bob sends only $\bar{0}$'s for the rest of the megablock, ignoring any instructions to send other messages. He still updates S_0, S_1, checks compatibility of S_b with m (Case 2), and transitions to Phase 2 or 3 if appropriate (Case 3).

Case 5: $\mathsf{cnt}_0 = \mathsf{cnt}_1 \neq 2i$ *and* $\mathsf{rec}_0 = \mathsf{rec}_1 = \mathsf{false}$.

Bob sends $\bar{1}$.

Case 6: $\mathsf{cnt}_0 = \mathsf{cnt}_1 \neq 2i$ *and* $\mathsf{rec}_0 \vee \mathsf{rec}_1 = \mathsf{true}$.

Bob sends $\bar{0}$.

At the end of the megablock, Bob begins the next megablock in Phase 1 again, unless he was specified to move to Phase 2 or 3.

Phase 2: Upon entering Phase 2 at the beginning of a megablock, Bob has previously decoded to two possible Alices in the previous megablock, at least one of which was in Stage 2 ($\mathsf{knt} \in \{0,1\}$). In fact, by Lemma 5.1,this is true for *exactly one* of the two Alices. That is, one Alice was in Stage 1 ($\mathsf{knt} = -1$) and the other was in Stage 2 ($\mathsf{knt} \in \{0,1\}$). W.l.o.g. suppose that the Stage 2 Alice corresponded to $\hat{x}_1$. Since Bob sent $\bar{1}$ in every message after this 2-decoding for the rest of the megablock, the real Alice must still be in the same stage as she was in when Bob 2-decoded. Now, Bob simply sends $\bar{0}$ in every message for the remainder of the protocol.

He continues list-decoding Alice's messages with $< \frac{3}{4} - \frac{3}{2}\epsilon$ erasures to $\{m_0, m_1\}$. If $S_b \cap \{m_0, m_1\} = \emptyset$ for some $b \in \{0,1\}$, he sets $\hat{x} \leftarrow \hat{x}_{1-b}$.

At the end of the protocol, let b be the bit Bob last received. He sets $\hat{x} \leftarrow \hat{x}_b$.

Phase 3: When Bob enters Phase 3, it is at the beginning of a megablock. He has previously list-decoded to two possible messages, at least one of which was of the form β^{4M} for some $\beta \in \{0,1\}$. Note that in fact exactly one of the messages must have been of the form β^{4M} since 0^{4M} and 1^{4M} have no overlapping bits. W.l.o.g. suppose that when Bob list-decoded to the two messages, the two messages were $m_0 = \mathsf{ECC}(\hat{x}_0, \mathsf{cnt}_0, \mathsf{cnfm}_0, \mathsf{rec}_0, \mathsf{knt}_0, \mathsf{stg2}_0) \in S_0$

and $m_1 = \beta_1^{4M} \in S_1$, where $\beta_1 \in \{0,1\}$. Bob's goal is to advance the Alice corresponding to m_0 to Stage 3 with the bit $\beta_0 = 1 - \beta_1$.

Since it is now the beginning of a megablock, the value of the relevant counter (either cnt_0 or knt_0) has been reset to 0. Let $\mathsf{counter}_0 = \mathsf{cnt}_0$ if $\mathsf{knt}_0 = -1$, and otherwise let $\mathsf{counter}_0 = \mathsf{knt}_0$.

In either case, the strategy is the same: In each megablock, Bob performs a similiar strategy as in Phase 1 to increment Alice's counter $\mathsf{counter}_0$ to a value j, and then send her $\bar{0}$'s for the rest of the megablock to advance her to Stage 3 with the bit $\beta_0 = 1 - \beta_1$. The value j is defined as follows: $j = 1 - \beta_1$ if $\mathsf{knt}_0 = -1$ and $j = \beta_1$ if $\mathsf{knt}_0 \in \{0,1\}$.

- If $j = 0$, Bob sends $\bar{0}$ for the rest of the protocol.
  He continues list-decoding Alice's messages with $< \frac{3}{4} - \frac{3}{2}\epsilon$ erasures to $\{m_0, m_1\}$. If $S_b \cap \{m_0, m_1\} = \emptyset$ for some $b \in \{0,1\}$, he sets $\hat{x} \leftarrow \hat{x}_{1-b}$.
- If $j = 1$, Bob does the following: he reads Alice's message m and sends a message based on the following cases.

  Case 1: *m has at least $\frac{3}{4} - \frac{3}{2}\epsilon$ $\perp$'s.*
  Bob sends the same message he sent previously, unless it's the start of a block, in which case he sends $\bar{1}$.

  Case 2: *Bob list decodes to ≤ 2 messages $\{m_0, m_1\}$, and $S_b \cap \{m_0, m_1\} = \emptyset$ for some $b \in \{0,1\}$. Note that $S_1 = \{\beta_1^{4M}\}$.*
  Bob sets $\hat{x} \leftarrow \hat{x}_{1-b}$.

  For the remaining cases, Bob can decode Alice's message into two options: $\mathsf{ECC}(x_0, \mathsf{cnt}_0, \mathsf{cnfm}_0, \mathsf{rec}_0, \mathsf{knt}_0, \mathsf{stg2}_0 = \mathsf{false})$ and β_1^{4M}.

  Case 3: $\mathsf{counter}_0 = 0$.
  Bob sends $\bar{1}$.

  Case 4: $\mathsf{counter}_0 = 1$.
  Bob sends $\bar{0}$ for the rest of the megablock.

At the end of the protocol, let b be the most recent bit received by Bob. He sets $\hat{x} \leftarrow \hat{x}_1$ if $b = \beta_1$ and $\hat{x} \leftarrow \hat{x}_0$ otherwise.

5.3 Analysis

5.3.1 Correctness Lemmas

We begin with a lemma that explains the correctness of our protocol. There are two parts: the first is that S_0 and S_1 can never overlap, that is, if the adversary is confusing Bob between two values of x, it is never possible that Alice's messages in the two worlds are the same. The second justifies an assumption we made in the protocol description, which is that if Bob 2-decodes Alice's message, then the two Alices cannot both be in Stage 2. We remind the reader that once Bob has set $\hat{x}$, he no longer updates (or initializes) the sets S_0, S_1.

Lemma 5.1. *At any point in Protocol 2 after S_0, S_1 are initialized,*

(i) $S_0 \cap S_1 = \emptyset$.

(ii) At most one of S_0, S_1 contains a message of the form $\mathsf{ECC}(x, \mathsf{cnt}, \mathsf{cnfm}, \mathsf{rec}, \mathsf{knt}, \mathsf{stg2})$ *with* $\mathsf{knt} \in \{0, 1\}$.

We delay the proof of Lemma 5.1 to the end of this section and first state a couple corollaries of the fact that $S_0 \cap S_1 = \emptyset$.

Corollary 5.2. *If it is ever the case that Bob uniquely decodes Alice's message, then Bob outputs x correctly. In particular, in order for Bob to output incorrectly, the adversary must erase at least half of each of Alice's messages.*

Proof. This is true the first time Alice's message to Bob has fewer than $\frac{3}{4} - \frac{3}{2}\epsilon$ erasures.

Otherwise, consider the first message m from Alice that Bob uniquely decodes to m'. By Lemma 5.1, $S_0 \cap S_1 = \emptyset$ so m' can only belong to one of S_0 and S_1. Since Alice's true message must always belong to the set S_b if her input is $\hat{x}_b$, this allows Bob to output x correctly by setting $\hat{x} \leftarrow \hat{x}_b$ where b is such that $m' \in S_b$. This is in fact what he does in the protocol. □

Corollary 5.3. *After $\hat{x}_0, \hat{x}_1, S_0, S_1$ are initialized, if Bob ever 2-decodes Alice's message to $\{m_0, m_1\}$ such that $S_b \cap \{m_0, m_1\} = \emptyset$ for some b, then Bob outputs x correctly.*

Proof. Alice's true message must always belong to the set S_b if her input is $\hat{x}_b$. Then, if $S_b \cap \{m_0, m_1\} = \emptyset$ for some b, it must be the case that $x = \hat{x}_{1-b}$. In the protocol, Bob in fact sets $\hat{x} \leftarrow \hat{x}_{1-b}$ if this is the case. □

Proof of Lemma 5.1. Assume for the sake of contradiction there exists an execution of the protocol in which at some point after S_0, S_1 are initialized, one of the conditions (i),(ii) is violated. Let V be the first chunk after which this is true, and let $W \leq V$ be the last chunk in which Alice's message to Bob was $< (\frac{3}{4} - \frac{3}{2}\epsilon)$-erased, so that Bob list-decoded Alice's message to two possibilities m_0 and m_1, where not both m_0 and m_1 are of the form β^{4M}. It must be that Bob has not set $\hat{x}$ as of chunk V, since as soon as he has set $\hat{x}$ he no longer updates (or initializes) S_0, S_1. It follows that either Alice's message was $< (\frac{3}{4} - \frac{3}{2}\epsilon)$-erased so that Bob list-decoded Alice's message to two possibilities (with $\hat{x}_0 \neq \hat{x}_1$ and $\mathsf{knt}_0 = \mathsf{knt}_1 = -1$) for the first time in chunk W, or $m_0 \in S_0$ and $m_1 \in S_1$, where S_0, S_1 are Bob's sets after chunk $W - 1$ that satisfy properties (i) and (ii). In either case, it holds that $m_0 \neq m_1$ and not both m_0, m_1 have $\mathsf{knt}_b \in \{0, 1\}$.

We will show that if $m_0 \neq m_1$ and not both have $\mathsf{knt}_b \in \{0, 1\}$, then properties (i) and (ii) hold for sets S_0 and S_1 after chunk V as well. We do so by casework on m_0, m_1.

> *Case 1:* $m_b = \mathsf{ECC}(\hat{x}_b, \mathsf{cnt}_b, \mathsf{cnfm}_b, \mathsf{rec}_b, \mathsf{knt}_b, \mathsf{stg2}_b)$ $\forall b \in \{0, 1\}$ *and* $\mathsf{knt}_0 = \mathsf{knt}_1 = -1$.
>
> > *Subcase 1.1:* $\mathsf{cnt}_0 = \mathsf{cnt}_1 < 2i$ *and* $\mathsf{rec}_0 = \mathsf{rec}_1 = \mathsf{true}$. *Bob has sent $\bar{0}$'s for the rest of the block since chunk W followed by $\bar{1}$'s in subsequent blocks.*
> >
> > Note that in neither world can Alice advance: if she hears a 0, she can at most confirm her current value of cnt. Then, all values in S_b after chunk V must still have $\mathsf{knt}_b = -1$ and $x_b = \hat{x}_b$, so that $S_0 \cap S_1 = \emptyset$ and no elements of either S_b have $\mathsf{knt} \in \{0, 1\}$.
>
> > *Subcase 1.2:* $\mathsf{cnt}_0 = \mathsf{cnt}_1 < 2i$ *and* $\mathsf{rec}_0 \neq \mathsf{rec}_1$. *Bob has sent $\bar{0}$'s for the rest of the block since chunk W followed by $\bar{1}$'s in subsequent blocks.*
> >
> > W.l.o.g. suppose $\mathsf{rec}_0 = \mathsf{true}$ and $\mathsf{rec}_1 = \mathsf{false}$. Note that since $\mathsf{rec}_0 = \mathsf{true}$ then $\mathsf{cnfm}_0 = \mathsf{false}$. Then S_0 consists only of messages with $\mathsf{knt} = -1$ and $x_0 = \hat{x}_0$,

as the corresponding Alice could only have confirmed her counter or incremented further. On the other hand, the set S_1 can only consist of states of the form β^{4M} or $\mathsf{ECC}(\hat{x}_1, \mathsf{cnt}_1, \mathsf{cnfm}_1, \mathsf{rec}_1, \mathsf{knt}_1, \mathsf{stg2}_1)$ with $\hat{x}_1 \neq \hat{x}_0$. Thus, $S_0 \cap S_1 = \emptyset$ and S_0 contains no messages with $\mathsf{knt} \in \{0, 1\}$.

Subcase 1.3: $\mathsf{cnt}_0 = \mathsf{cnt}_1 < 2i$ *and* $\mathsf{rec}_0 = \mathsf{rec}_1 = \mathsf{false}$. *Bob has sent* $\bar{1}$*'s for the rest of the block since chunk* W *and in subsequent blocks.*

Neither state could have advanced, since no 0's were sent between chunks W and V. This means that S_b consists of only elements with $x_b = \hat{x}_b$ and $\mathsf{knt} = -1$, where $\hat{x}_0 \neq \hat{x}_1$, so $S_0 \cap S_1 = \emptyset$ and neither S_b contains a message with $\mathsf{knt} \in \{0, 1\}$.

Subcase 1.4: $\mathsf{cnt}_0 \neq \mathsf{cnt}_1$ *or* $\mathsf{cnt}_0 = \mathsf{cnt}_1 = 2i$. *Bob has sent* $\bar{0}$*'s for the rest of the megablock since chunk* W*, and* $\bar{1}$*'s in subsequent megablocks.*

We are only concerned with the elements of S_0, S_1 that do not have $\mathsf{knt} = -1$. That is, the interesting elements are those that Alice would send had she heard a $\bar{0}$ from Bob in the megablock containing chunk W.

First, at most one of the sets S_0, S_1 can contain an element with $\mathsf{knt} \in \{0, 1\}$. This is because Bob begins sending $\bar{0}$'s the *first time* he sees either $\mathsf{cnt}_0 \neq \mathsf{cnt}_1$ or $\mathsf{cnt}_0 = \mathsf{cnt}_1 = 2i$, meaning that either $\mathsf{cnt}_0 = \mathsf{cnt}_1 \pm 1$ or $\mathsf{cnt}_0 = \mathsf{cnt}_1 = 2i$, where $\hat{x}_0[i] \neq \hat{x}_1[i]$. It's thus not possible that both Alices satisfy the condition for advancing to Stage 2, namely, $\mathsf{cnt}_b \bmod 2 = 0$ and $\hat{x}_b[\mathsf{cnt}_b/2] = 1$.

Second, the sets S_0, S_1 cannot contain the same element β^{4M}. This is because Alice only advances to Stage 3 if $\mathsf{cnt} \bmod 2 = 0$ and $x[\mathsf{cnt}/2] = 0$ (with $\beta = 0$), or $\mathsf{cnt} \bmod 2 = 1$ (with $\beta = 1$). Since either $\mathsf{cnt}_0 = \mathsf{cnt}_1 \pm 1$ or $\mathsf{cnt}_0 = \mathsf{cnt}_1 = 2i$, where $\hat{x}_0[i] \neq \hat{x}_1[i]$, the two Alices cannot both have $\mathsf{cnt} \bmod 2 = 0$ and $x[\mathsf{cnt}/2] = 0$ or both have $\mathsf{cnt} \bmod 2 = 1$.

Case 2: One of m_0, m_1 *is of the form* β^{4M}.

W.l.o.g. suppose $m_1 = \beta_1^{4M}$ and $m_0 = \mathsf{ECC}(\hat{x}_0, \mathsf{cnt}_0, \mathsf{cnfm}_0, \mathsf{rec}_0, \mathsf{knt}_0, \mathsf{stg2}_0)$. Then $S_1 = \{\beta_1^{4M}\}$. We will show that $\beta_1^{4M} \notin S_0$.

If Bob has only sent $\bar{1}$'s to Alice since chunk W, then S_0 contains no elements of the form β^{4M} as Alice cannot advance without receiving a $\bar{0}$.

Bob has only sent a positive number of $\bar{0}$'s since chunk W if either $j = 1$ and $\mathsf{counter}_0 = 1$ (in which case Bob sends $\bar{0}$'s for the rest of the megablock containing W) or $j = 0$ and V is in a different megablock than W (in which case Bob sends $\bar{0}$ only in the megablocks after chunk W). Recall that j is chosen such that Alice can only advance to Stage 3 with the bit $1 - \beta_1$ if $\mathsf{counter}_0 = j$. Furthermore, Bob only sends $\bar{0}$'s when he knows that Alice's value of $\mathsf{counter}_0$ is equal to j. Thus, $\beta_1^{4M} \notin S_0$.

Case 3: One of m_0, m_1 *has* $\mathsf{knt} = -1$ *while the other has* $\mathsf{knt} \in \{0, 1\}$.

W.l.o.g. suppose m_0 has $\mathsf{knt}_0 = -1$ and m_1 has $\mathsf{knt}_1 \in \{0, 1\}$. Recall also that in order for Alice to be in Stage 2, it must be the case that $\mathsf{cnt}_1 \bmod 2 = 0$ and $\hat{x}_1[\mathsf{cnt}_1/2] = 1$.

It is impossible that S_0 contains an element with $\mathsf{knt} \in \{0, 1\}$, as Alice doesn't advance unless she receives a $\bar{0}$ and Bob only sends $\bar{0}$'s when he's sure that Alice's $\mathsf{counter}_0 = j$, for which Alice would advance directly to Stage 3. Therefore, S_0 and S_1 do not both have an element with $\mathsf{knt} \in \{0, 1\}$.

We next show that S_0 and S_1 cannot contain the same element β^{4M}. If V is in the same megablock as the first time Bob list-decoded to two messages with $\mathsf{knt}_0 = -1$ and $\mathsf{knt}_1 \in \{0,1\}$, then Bob has only sent $\bar{1}$'s since then and in particular since chunk W. Alice cannot advance without receiving a $\bar{0}$, so that S_0 and S_1 in fact contain no elements of the form β^{4M}. Otherwise, Bob has only sent $\bar{0}$'s after that first megablock. Recall since Alice resets $\mathsf{counter}_b$ to 0 at the beginning of megablocks, she could only have received a $\bar{0}$ if $\mathsf{counter}_b = 0$. Then S_0 can contain the element 0^{4M} (but not 1^{4M}) and S_1 can contain the element 1^{4M} (but not 0^{4M}). Thus $S_0 \cap S_1 = \emptyset$.

□

5.3.2 Main Theorem: 3/5 Erasure Resilience

We now prove our main theorem that Protocol 2 is resilient to $\frac{3}{5} - O(\epsilon)$ adversarial erasures. First, we establish a list of lemmas that we will use in the proof of Theorem 5.7.

Lemma 5.4. *Let R be the megablock in which Alice advances to Stage 3, or $R = A$ if she never advances to Stage 3. If Bob is in Phase 2 or 3 at the end of the protocol, and if he receives any bit sent after megablock R, then Bob outputs x correctly.*

Proof. If Bob is in Phase 2 at the end of the protocol, this means that at some point the two list-decoded states are such that $\mathsf{knt}_0 = -1$ and $\mathsf{knt}_1 \in \{0,1\}$. When Bob transitions to Phase 2 at the start of a megablock, it must still hold that $\mathsf{knt}_0 = -1$ for all $m_0 \in S_0$ and $\mathsf{knt}_1 \in \{0,1\}$ for all $m_1 \in S_1$, since Bob only sent $\bar{1}$ for the rest of the previous megablock so that Alice could not further advance. In other words, if the real Alice has input $x = \hat{x}_0$, she must still be in Stage 1, and if she has $x = \hat{x}_1$, she must be in Stage 2. Bob then sends $\bar{0}$ for the rest of the protocol. If the real Alice had $x = \hat{x}_0$ and were in Stage 1, when she receives a bit from Bob, she advances directly to Stage 3 with the bit $B = 0$. If she had $x = \hat{x}_1$ and were in Stage 2, when she receives a bit from Bob, she advances to Stage 3 with $B = 1$. Thus, if Alice advanced and any subsequent bit is received by Bob, he outputs x correctly.

If Bob is in Phase 3 at the end of the protocol, this means that he at some point list-decoded Alice's message to two messages, one of the form B^{4M} and the other of the form $\mathsf{ECC}(\hat{x}_b, \mathsf{cnt}_b, \mathsf{cnfm}_b, \mathsf{rec}_b, \mathsf{knt}_b, \mathsf{stg2}_b)$. If Alice's real input were $\hat{x}_b$, note that by design, she can only advance to Stage 3 with the bit $1 - B$. Thus, if Alice advanced to Stage 3 and at least one subsequent bit was received by Bob, he outputs x correctly. □

Lemma 5.5. *If Bob begins a megablock in Phase 1 and does not transition to Phase 2 or 3 at the end of the megablock, nor has he set $\hat{x}$ by the end of the megablock, the adversary must have erased at least $\frac{3}{5} - \frac{18}{5}\epsilon$ of the megablock.*

Proof. Let E be the last block where fewer than $\frac{3}{4} - \frac{3}{2}\epsilon$ of Alice's bits were erased. Note that if no such block exists, then the adversary trivially erased at least

$$\frac{\left(\frac{3}{4} - \frac{3}{2}\epsilon\right) \cdot B \cdot C \cdot 4M}{B \cdot C \cdot 5M} = \frac{3}{5} - \frac{6}{5}\epsilon > \frac{3}{5} - \frac{18}{5}\epsilon$$

of the megablock, so we may assume such a block exists. Within this block, there is a chunk in which Alice's message is $< (\frac{3}{4} - \frac{3}{2}\epsilon)$-erased, so that Bob list-decoded her message to two messages,

both of which must have $\mathsf{knt} = -1$ as Bob does not transition to Phase 2 or 3 at the end of the megablock. This means that as of right before block E, both the real and the fake Alice are still in Stage 1.

Since Alice never increments past $2i$, before block E, there were at most $2i \leq 2n$ blocks in which Alice heard at least one $\bar{0}$ after hearing some nonzero number of $\bar{1}$'s. In the other $\geq (E-1) - 2n$ blocks, Alice heard only $\bar{1}$'s and erased messages. In such blocks, let V be the first chunk in which Bob decodes Alice's message to two possibilities, one in S_0 and one in S_1 such that $\mathsf{rec}_0 \vee \mathsf{rec}_1 = \mathsf{true}$, so that he sends $\bar{1}$'s before chunk V and $\bar{0}$'s in or after chunk V. Since Alice only hears $\bar{1}$'s, she does not receive any of Bob's messages in or after chunk V. Furthermore, since chunk V was the first chunk in which Bob list-decodes Alice's message to two messages such that at least one value of rec is true, it must be the case that in all but one chunk before V, the adversary erased either $\frac{3}{4} - \frac{3}{2}\epsilon$ of Alice's message or all of Bob's and $\frac{1}{2} - \epsilon$ of Alice's. To summarize, in each of the $\geq (E-1) - 2n$ blocks, at least

$$\begin{aligned}
&\frac{(V-2) \cdot \min\left\{(\tfrac{3}{4} - \tfrac{3}{2}\epsilon) \cdot 4M, (\tfrac{1}{2} - \epsilon) \cdot 4M + M\right\} + (C - V + 1) \cdot \left((\tfrac{1}{2} - \epsilon) \cdot 4M + M\right)}{C \cdot 5M} \\
&\geq \left(\frac{3}{5} - \frac{6}{5}\epsilon\right) \cdot \frac{C-1}{C} \\
&\geq \left(\frac{3}{5} - \frac{6}{5}\epsilon\right) \cdot (1 - \epsilon) \\
&\geq \frac{3}{5} - \frac{9}{5}\epsilon
\end{aligned}$$

of the rounds were erased, where we used that $C = \lceil \frac{1}{\epsilon} \rceil$. Combining this with the fact that after block E, $\frac{3}{4} - \frac{3}{2}\epsilon$ of each of Alice's messages is erased, we get that

$$\begin{aligned}
&\geq \frac{(E - 1 - 2n) \cdot \left(\frac{3}{5} - \frac{9}{5}\epsilon\right) \cdot C \cdot 5M + (B - E) \cdot \left(\frac{3}{4} - \frac{3}{2}\epsilon\right) \cdot C \cdot 4M}{B \cdot C \cdot 5M} \\
&= \frac{(\frac{3}{5} - \frac{6}{5}\epsilon) \cdot B - \frac{3}{5}\epsilon E - (\frac{3}{5} - \frac{9}{5}\epsilon) \cdot (2n+1)}{B} \\
&\geq \left(\frac{3}{5} - \frac{9}{5}\epsilon\right)\left(1 - \frac{2n+1}{B}\right) \\
&\geq \left(\frac{3}{5} - \frac{9}{5}\epsilon\right)(1 - 3\epsilon) \\
&\geq \frac{3}{5} - \frac{18}{5}\epsilon
\end{aligned}$$

of the megablock must have been erased, where we used that $B = \lceil \frac{n}{\epsilon} \rceil$. □

Lemma 5.6. *If Bob begins a megablock in Phase 2 or 3, and Alice is not in Stage 3 nor has Bob set $\hat{x}$ by the end of the megablock, the adversary must have corrupted at least $\frac{3}{5} - \frac{12}{5}\epsilon$ of the megablock.*

Proof. If Bob begins the megablock in Phase 2, he sends $\bar{0}$'s for the entire megablock. Note that Alice has reset the value of her $\mathsf{counter}$ to 0, so that upon receiving a $\bar{0}$ from Bob she advances straight to Stage 3. Thus, in order for Alice not to advance to Stage 3 in this megablock, the adversary must erase all of Bob's messages. They must also erase at least $\frac{1}{2} - \epsilon$ of each of Alice's messages, or by Lemma 5.2 Bob correctly sets $\hat{x}$ to x.

Thus, the adversary must corrupt a total of

$$\geq \frac{\left(\frac{1}{2}-\epsilon\right)\cdot B\cdot C\cdot 4M + B\cdot C\cdot M}{B\cdot C\cdot 5M} \geq \frac{3}{5}-\frac{4}{5}\epsilon > \frac{3}{5}-\frac{12}{5}\epsilon$$

of the megablock.

If Bob begins the megablock in Phase 3 and Alice does not advance to Stage 3, there are $j \leq 1$ blocks in which Alice hears a $\bar{0}$. In the other $\geq B-1$ blocks, Alice hears only 1's. Then, in each of these $\geq B-1$ blocks, as in the proof of Lemma 5.5, $\geq \frac{3}{5}-\frac{9}{5}\epsilon$ of the rounds must have been erased.

As such, we get that

$$\begin{aligned}
&\geq \frac{(B-1)\cdot(\frac{3}{5}-\frac{9}{5}\epsilon)\cdot C\cdot 5M}{B\cdot C\cdot 5M}\\
&\geq \left(\frac{3}{5}-\frac{9}{5}\epsilon\right)\cdot\left(1-\frac{1}{B}\right)\\
&\geq \left(\frac{3}{5}-\frac{9}{5}\epsilon\right)\cdot(1-\epsilon)\\
&\geq \frac{3}{5}-\frac{12}{5}\epsilon
\end{aligned}$$

of the megablock must have been erased. □

We are now ready to prove that Protocol 2 is resilient to $\frac{3}{5}-\frac{24}{5}\epsilon$ erasures.

Theorem 5.7. *Protocol 2 is resilient to a $\frac{3}{5}-\frac{24}{5}\epsilon$ fraction of erasures. For an input of size n, the total communication is $O_\epsilon(n^2)$. Alice and Bob run in time $\text{poly}_\epsilon(n)$.*

Proof. The communication complexity is $A\cdot B\cdot C\cdot 5M = \lceil\frac{1}{\epsilon}\rceil\cdot\lceil\frac{n}{\epsilon}\rceil\cdot\lceil\frac{1}{\epsilon}\rceil\cdot O_\epsilon(n) = O_\epsilon(n^2)$. The runtimes of Alice and Bob is governed by the time it takes to list-decode Alice's messages, which is $\text{poly}_\epsilon(n)$ by Theorem 3.4. We focus on proving error resilience.

Suppose that the adversary erases some number of rounds such that Bob outputs the incorrect value of x at the end of the protocol. This in particular means that Bob never uniquely decodes Alice's message, nor does he decode to two messages, neither of which are in the set S_b for some $b\in\{0,1\}$, otherwise by Lemmas 5.2 and 5.3 Bob sets $\hat{x}\leftarrow x$ correctly. We will show that there must have been at least $\frac{3}{5}-\frac{24}{5}\epsilon$ erasures.

Let $R\in[A]$ be the megablock in which Alice advances to Stage 3; if she never advances to Stage 3, let $R=A$. If Bob ends the protocol in Phase 1, it must be the case that each of Alice's messages after the R'th megablock were $(\frac{3}{4}-\frac{3}{2}\epsilon)$-erased, otherwise Bob would transition from Phase 1 to Phase 3 upon seeing one world with $\mathsf{knt}=\infty$. If Bob is in Phase 2 or 3 at the end of the protocol, Bob couldn't have seen any bit from Alice sent after the R'th megablock, otherwise by Lemma 5.4 he correctly outputs x. In either case, it is the case that all messages from Alice after the R'th megablock are at least $(\frac{3}{4}-\frac{3}{2}\epsilon)$-erased.

We claim that there is at most one megablock before megablock R with fewer than $\frac{3}{5}-\frac{18}{5}$ erasures. In the first such block, Bob must transition from Phase 1 to Phase 2 or 3 by Lemma 5.5. In the second block with fewer than $\frac{3}{5}-\frac{18}{5}\epsilon$ erasures, by Lemma 5.6 Alice must have advanced to Stage 3 by its end. Thus, this second megablock cannot be before megablock R, as Alice advances to Stage 3 in megablock R (or not at all). Therefore, in each of $R-2$ of the first $R-1$ megablocks,

at least $\frac{3}{5} - \frac{18}{5}$ of the megablock is erased. Combining this with the fact that in each of the final $A - R$ megablocks, Alice's messages are at least $(\frac{3}{4} - \frac{3}{2}\epsilon)$-erased, we get that

$$\begin{aligned}
&\geq \frac{(R-2)\cdot(\frac{3}{5} - \frac{18}{5}\epsilon)\cdot B\cdot C\cdot 5M + (A-R)\cdot(\frac{3}{4} - \frac{3}{2}\epsilon)\cdot B\cdot C\cdot 4M}{A\cdot B\cdot C\cdot 5M} \\
&= \frac{(\frac{3}{5} - \frac{6}{5}\epsilon)A - \frac{12}{5}\epsilon R - (\frac{6}{5} - \frac{36}{5}\epsilon)}{A} \\
&\geq \frac{3}{5} - \frac{6}{5}\epsilon - \frac{12}{5}\epsilon - \frac{6/5}{A} \\
&\geq \frac{3}{5} - \frac{6}{5}\epsilon - \frac{6}{5}\epsilon \\
&= \frac{3}{5} - \frac{24}{5}\epsilon
\end{aligned}$$

of the rounds in the protocol must have been erased, where we use that $R \leq A$ and $A = \lceil \frac{1}{\epsilon} \rceil$. □

6 Impossibility Bounds

6.1 Binary Erasure Channel

Theorem 6.1. *No* iECC *is resilient to a* $\frac{2}{3}$ *fraction of erasures with probability greater than* $\frac{1}{2}$.

Proof. Consider the frequency with which Bob speaks. If he speaks $r \leq \frac{1}{3}$ of the time, consider the following attack: The adversary erases all of Bob's messages. Then, Alice's messages to Bob reduce to an error correcting code, which has relative distance at most $\frac{1}{2}$. Thus, the adversary can erase half of Alice's messages such that Bob cannot distinguish between two of Alice's inputs. This attack uses $r + \frac{1-r}{2} = \frac{1+r}{2} \leq \frac{2}{3}$ erasures.

Alternatively, if Bob speaks $r > \frac{1}{3}$ of the time, the adversary can simply erase all of Alice's messages to Bob. Then, Bob receives no information about Alice's input. This attack requires $< \frac{2}{3}$ erasures.

Thus, there is always an attack using $\leq \frac{2}{3}$ erasures for which Bob cannot correctly output Alice's input with probability greater than $\frac{1}{2}$. □

6.2 Binary Bit Flip Channel

We also propose the problem of constructing iECCs in the standard bit flip corruption model. It is trivial to construct an iECC resilient to $\frac{1}{4}$ adversarial bit flips: simply have Alice send ECC(x), where ECC is a standard error correcting code of relative distance $\frac{1}{2}$. The following is a corollary of Theorem 3.2.

Proposition 6.2. *For any* $\epsilon > 0$*, there exists an* iECC *resilient to* $\frac{1}{4} - \epsilon$ *adversarial bit flips such that the communication complexity for inputs of size* n *is* $O_\epsilon(n)$.

Without interaction, one cannot achieve error resilience greater than $\frac{1}{4}$. Unfortunately, we do not know of any protocols that use interaction to achieve a higher error resilience. We leave constructing one an open problem.

Instead, we prove the following impossibility bound on the error resilience of any iECC in the bit flip setting.

Theorem 6.3. *No* iECC *is resilient to more than* $\frac{2}{7}$ *errors with probability greater than* $\frac{1}{2}$.

In our proof, we use the following classical result.

Theorem 6.4. *[SW92] No* iECC *is resilient to an attack consisting of corrupting* $\frac{1}{3}$ *of Alice's messages and none of Bob's messages with probability greater than* $\frac{1}{2}$.

Proof of Theorem 6.3. Let us first assume that Alice talks at most $\frac{6}{7}$ of the time. By Theorem 6.4, even if none of Bob's messages are corrupted, the adversary has a strategy to confuse Bob between two inputs by corrupting $\frac{1}{3}$ of the bits Alice sends. This attack requires results at most $\frac{1}{3} \cdot \frac{6}{7} = \frac{2}{7}$ corruptions.

Now, let us assume that Alice talks at least $\frac{6}{7}$ of the time. Suppose that Alice talks in A rounds and Bob talks in B rounds. We can divide the protocol into C chunks in which Alice talks contiguously for some number of rounds then Bob speaks contiguously for some number of rounds (the number of rounds within each chunk does not have to be the same). Let Alice's possible inputs be $x_1, \ldots, x_N$. For $k \in \{1 \ldots C\}$, we define $A_k(x_i)$ and $B_k(x_i, x_j)$ and construct $R_k(x_i, x_j)$ and S_k as follows:

- For each $i \in [N]$, let $A_k(x_i)$ be what Alice would send in the k'th chunk if she has input x_i and she's seen the $k-1$ messages $S_1, \ldots, S_{k-1}$ from Bob. (If $k = 1$ then $A_k(x_i)$ is simply what Alice would send in the first chunk if her input were x_i).
- For each pair $i \neq j$, let $R_k(x_i, x_j)$ be a string that's minimally equidistant from $A_k(x_i)$ and $A_k(x_j)$, satisfying

$$\Delta(R_k(x_i, x_j), A_k(x_i)) = \Delta(R_k(x_i, x_j), A_k(x_j)) = \frac{1}{2} \cdot \Delta(A_k(x_i), A_k(x_j)),$$

  This can be constructed by letting $R_1(x_i, x_j)[\ell] = A_1(x_i)[\ell] = A_1(x_j)[\ell]$ when the ℓ'th bit of $A_1(x_i)$ and $A_1(x_j)$ are the same, and $R_1(x_i, x_j)[\ell] = A_1(x_i)[\ell]$ half the time and $R_1(x_i, x_j)[\ell] = A_1(x_j)$ half the time when $A_1(x_i)[\ell] \neq A_1(x_j)[\ell]$.
- For each $i \neq j$, let $B_k(x_i, x_j)$ be Bob's message if he's seen the messages $R_1(x_i, x_j), \ldots, R_k(x_i, x_j)$ from Alice so far.
- Define S_k bitwise by setting $S_k[\ell] = \text{maj}\{B_k(x_i, x_j)\}_{i,j}$.

Now, for all pairs $i \neq j$, define the transcript τ_{ij} to be where Bob receives $R_k(x_i, x_j)$ and Alice receives S_k in each chunk $k \in [C]$. We will show that for some $i \neq j$, in both the case Alice has x_i or she has x_j it costs few corruptions for an adversary to corrupt the protocol so that the resulting communication is τ_{ij}. Note that the number of corruptions needed so that the communication is τ_{ij} is the same whether Alice has x_i or x_j, since Bob's message is corrupted identically in both cases and $R_k(x_i, x_j)$ is picked to be equidistant from $A_k(x_i)$ and $A_k(x_j)$.

Consider this attack on the protocol where Alice has x_i and the resulting transcript is τ_{ij}. The

total number of corruptions over all (ordered) pairs is:

$$\begin{aligned}
&\sum_{ij}\sum_{k}\Big[\Delta\big(S_k, B_k(x_i,x_j)\big) + \Delta\big(R_k(x_i,x_j), A_k(x_i)\big)\Big] \\
&= \sum_{ijk}\left[\Delta\big(S_k, B_k(x_i,x_j)\big) + \frac{1}{2}\cdot\Delta\big(A_k(x_i), A_k(x_j)\big)\right] \\
&\qquad\qquad \le \frac{1}{2}B\cdot N(N-1) + \frac{1}{2}\cdot A\cdot\frac{1}{2}N^2 \\
&\qquad\qquad \le \left(\frac{1}{2}B + \frac{1}{4}A\right)N^2,
\end{aligned}$$

where the inequality $\sum_{ijk}\Delta(S_k, B_k(x_i,x_j)) \le \frac{1}{2}B\cdot N(N-1)$ follows from the fact that for each bit $j\in[B]$, $S_k[j]$ is different from at most half of the values of $B_k(x_i,x_j)[j]$, and $\Delta(A_k(x_i),A_k(x_j)) \le A\cdot\frac{1}{2}N^2$ follows from the fact that for any position $j\in[A]$, across the N codewords $A_k(x_i)$, there are at most $\frac{1}{2}N^2$ pairs of codewords with different values for bit j.

Therefore, by the pigeonhole principle, there is some pair i,j for which the attack uses $\le \frac{1}{2}B + \frac{1}{4}A \le \frac{1}{2}\cdot\frac{1}{7} + \frac{1}{4}\cdot\frac{6}{7} = \frac{2}{7}$ corruptions.

□

7 Acknowledgments

We would like to thank Venkat Guruswami, Yury Polyanskiy, and Raghuvansh Saxena for enlightning comments and helpful discussions. We would also like to thank Yang P. Liu and Mark Sellke for reading the paper and providing feedback.

Finally, we want to thank Aliceurill, Bobasaur, and Eevee, without whom this paper would not be possible. As seen in Figure 3, the three remain good friends despite Eevee's occasional adversarial tendencies.

References

[ADL06] Rudolf Ahlswede, Christian Deppe, and Vladimir Lebedev. Non-binary error correcting codes with noiseless feedback, localized errors, or both. In *2006 IEEE International Symposium on Information Theory*, pages 2486–2487, 2006. 3

[BE14] M. Braverman and K. Efremenko. List and unique coding for interactive communication in the presence of adversarial noise. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 236–245, Los Alamitos, CA, USA, oct 2014. IEEE Computer Society. 3

[Ber64] Elwyn R. Berlekamp. Block coding with noiseless feedback. 1964. 1, 2

[Ber68] Elwyn R. Berlekamp. Block coding for the binary symmetric channel with noiseless, delayless feedback. *Error-correcting Codes*, pages 61–88, 1968. 3

[BK12] Zvika Brakerski and Yael Tauman Kalai. Efficient interactive coding against adversarial noise. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 160–166, 2012. 3

Figure 3: (Left to right) Eevee, Aliceurill, and Bobasaur

[BN13] Zvika Brakerski and Moni Naor. Fast algorithms for interactive coding. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, page 443–456, USA, 2013. Society for Industrial and Applied Mathematics. 3

[BR11] Mark Braverman and Anup Rao. Towards coding for maximum errors in interactive communication. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, STOC '11, page 159–166, New York, NY, USA, 2011. Association for Computing Machinery. 3

[Bra12] Mark Braverman. Towards deterministic tree code constructions. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 161–167, New York, NY, USA, 2012. Association for Computing Machinery. 3

[BY08a] Marat Burnashev and Hirosuke Yamamoto. On the zero-rate error exponent for a bsc with noisy feedback. *Problems of Information Transmission*, 44, 09 2008. 3

[BY08b] Marat V. Burnashev and Hirosuke Yamamoto. On bsc, noisy feedback and three messages. In *2008 IEEE International Symposium on Information Theory*, pages 886–889, 2008. 3

[EGH16] Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Maximal noise in interactive communication over erasure channels and channels with feedback. *IEEE Trans. Inf. Theory*, 62(8):4575–4588, 2016. 3

[EKS20] Klim Efremenko, Gillat Kol, and Raghuvansh R. Saxena. Binary interactive error resilience beyond $1/8$ (or why $(1/2)^3 > 1/8$). In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 470–481, 2020. 3, 4

[FGOS15] Matthew Franklin, Ran Gelles, Rafail Ostrovsky, and Leonard J. Schulman. Optimal coding for streaming authentication and interactive communication. *IEEE Transactions on Information Theory*, 61(1):133–145, 2015. 3

[Gel17] Ran Gelles. Coding for interactive communication: A survey. *Foundations and Trends® in Theoretical Computer Science*, 13:1–161, 01 2017. 3

[GH13] Mohsen Ghaffari and Bernhard Haeupler. Optimal error rates for interactive coding ii: Efficiency and list decoding. *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, 12 2013. 3, 4

[GH17] Ran Gelles and Bernhard Haeupler. Capacity of interactive communication over erasure channels and channels with feedback. *SIAM Journal on Computing*, 46:1449–1472, 01 2017. 3

[GHK+16] Ran Gelles, Bernhard Haeupler, Gillat Kol, Noga Ron-Zewi, and Avi Wigderson. *Towards Optimal Deterministic Coding for Interactive Communication*, pages 1922–1936. 2016. 3

[GS00] Venkatesan Guruswami and Madhu Sudan. List decoding algorithms for certain concatenated codes. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC '00, page 181–190, New York, NY, USA, 2000. Association for Computing Machinery. 11, 12

[Gur03] V. Guruswami. List decoding from erasures: bounds and code constructions. *IEEE Transactions on Information Theory*, 49(11):2826–2833, 2003. 12

[GZ21] Meghal Gupta and Rachel Yun Zhang. The optimal error resilience of interactive communication over binary channels, 2021. 3

[Hae14] Bernhard Haeupler. Interactive channel capacity revisited. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 226–235, 2014. 3

[Ham50] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950. 1

[HKV15] Bernhard Haeupler, Pritish Kamath, and Ameya Velingker. Communication with partial noiseless feedback. In *APPROX-RANDOM*, 2015. 3

[Sch92] L.J. Schulman. Communication on noisy channels: a coding theorem for computation. In *Proceedings., 33rd Annual Symposium on Foundations of Computer Science*, pages 724–733, 1992. 3

[Sch93] Leonard J. Schulman. Deterministic coding for interactive communication. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, page 747–756, New York, NY, USA, 1993. Association for Computing Machinery. 3

[Sch96] Leonard J Schulman. Coding for interactive communication. *IEEE Transactions on Information Theory*, 42(6):1745–1756, 1996. 3

[Sha48] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. 1

[SW92] Joel Spencer and Peter Winkler. Three thresholds for a liar. *Combinatorics, Probability and Computing*, 1(1):81–93, 1992. 3, 30

[WQC17] Gang Wang, Yanyuan Qin, and Chengjuan Chang. Communication with partial noisy feedback. In *2017 IEEE Symposium on Computers and Communications (ISCC)*, pages 602–607, 2017. 1, 3

[Zig76] K.Sh. Zigangirov. Number of correctable errors for transmission over a binary symmetrical channel with feedback. *Problems Inform. Transmission*, 12:85–97, 1976. 3