Computation over the Noisy Broadcast Channel with Malicious Parties

Klim Efremenko
Ben Gurion University of the Negev, Beer Sheva, Israel
klimefrem@gmail.com

Gillat Kol
Princeton University, NJ, USA
gillat.kol@gmail.com

Dmitry Paramonov
Princeton University, NJ, USA
dp20@cs.princeton.edu

Raghuvansh R. Saxena
Princeton University, NJ, USA
rrsaxena@princeton.edu

Abstract

We study the n-party *noisy broadcast channel* with a constant fraction of *malicious parties*. Specifically, we assume that each non-malicious party holds an input bit, and communicates with the others in order to learn the input bits of all non-malicious parties. In each communication round, one of the parties broadcasts a bit to all other parties, and the bit received by each party is flipped with a fixed constant probability (independently for each recipient). How many rounds are needed?

Assuming there are no malicious parties, Gallager gave an $\mathcal{O}(n \log\log n)$-round protocol for the above problem, which was later shown to be optimal. This protocol, however, inherently breaks down in the presence of malicious parties.

We present a novel $n \cdot \tilde{\mathcal{O}}\left(\sqrt{\log n}\right)$-round protocol, that solves this problem even when almost half of the parties are malicious. Our protocol uses a new type of error correcting code, which we call a *locality sensitive code* and which may be of independent interest. Roughly speaking, these codes map "close" messages to "close" codewords, while messages that are not close are mapped to codewords that are very far apart.

We view our result as a first step towards a theory of *property preserving interactive coding*, i.e., interactive codes that preserve useful properties of the protocol being encoded. In our case, the naive protocol over the noiseless broadcast channel, where all the parties broadcast their input bit and output all the bits received, works even in the presence of malicious parties. Our simulation of this protocol, unlike Gallager's, preserves this property of the original protocol.

2012 ACM Subject Classification Theory of computation → Communication complexity

Keywords and phrases Broadcast Network, Malicious Parties, Communication Complexity

Digital Object Identifier 10.4230/LIPIcs.ITCS.2021.82

Related Version `https://eccc.weizmann.ac.il/report/2021/001/`

Funding *Klim Efremenko*: Supported by the Israel Science Foundation (ISF) through grant No. 1456/18 and European Research Council Grant number: 949707.
Gillat Kol: Supported by an Alfred P. Sloan Fellowship, the National Science Foundation CAREER award CCF-1750443, and by the E. Lawrence Keyes Jr. / Emerson Electric Co. Award.
Dmitry Paramonov: Supported by the National Science Foundation CAREER award CCF-1750443.
Raghuvansh R. Saxena: Supported by the National Science Foundation CAREER award CCF-1750443.

Acknowledgements We thank Huacheng Yu for helpful discussions.

1 Introduction

The field of *interactive coding*, pioneered by Schulman [16, 17, 18], asks the following question:

> *Let* Π *be a communication protocol designed to work over a noiseless channel. Can* Π *be converted to a noise resilient protocol* Π' *with similar communication complexity?*

Many works, mainly over the last decade, give affirmative answers to this question for the two-party channel, as well as various multi-party distributed channels. For example, it was shown that protocols in the extensively studied message passing model (peer-to-peer channels) and in the shared blackboard model (broadcast channel), can be *simulated* by protocols that tolerate stochastic noise, i.e., noise that flips each of the communicated bits with constant probability [15, 1, 4, 7, 12, 6, 13, 5]. These only incur a small (sub-logarithmic and in many cases, even constant) multiplicative overhead to the communication. Here, by "simulate" we mean that the new protocols retain the same input-output behavior as the original protocols[1].

Property preserving interactive coding. While the simulation protocols Π', designed by the aforementioned interactive coding works, are communication efficient and preserve the input-output behavior of the original protocols Π, they often lose the "structure" of the original protocols together with some of the basic properties making the original protocols useful. For instance, the importance of celebrated distributed protocols for the *consensus* and the *leader election* problems stems from their fault tolerance properties – the fact that they keep the same input-output behavior, even in the presence of *malicious parties* that may exhibit crashes or even Byzantine failures[2].

We study the above interactive coding question in a different light: Assume that the original communication protocol Π satisfies a special property P, can Π be converted to a noise resilient protocol Π' that still satisfies P? Specifically, we focus on protocols with the property $P =$ "Π is resilient to a constant fraction of malicious parties", and give a simulation protocol Π' over a noisy channel that also satisfies P.

The noisy broadcast channel. In this paper, we consider this new "property preserving" interactive coding question in the *noisy broadcast model*, a noisy version of the shared blackboard model, first suggested by El Gamal in 1984 [8]. In this model, a set of n parties, each holding a private input, communicate over a noisy broadcast channel. In each round, one of the parties broadcasts a bit to all the other parties, and the bit received by each of the other parties is flipped with some constant probability $\epsilon > 0$ (independently for each recipient).

We revisit the basic problem suggested by El Gamal, regarding the computation of the *identity function* over the noisy broadcast channel: assume that each party receives a single bit as an input and that the parties' mutual goal is for all parties to learn all input bits. That is, party i gets a bit $x^i \in \{0,1\}$ and needs to output $f(x^1, x^2, \cdots, x^n) = (x^1, x^2, \cdots, x^n)$. How many communication rounds are needed? Observe that, over the noiseless broadcast

[1] The parties participating in the distributed protocol are assumed to each have an input at the beginning of the protocol and give an output when the protocol terminates. We often think of the entire transcript received by a party as its output.

[2] Recall, for example, that in the consensus problem, each party gets an input bit and all parties need to output the input bit of one of the parties (and, in particular, all parties need to output the same bit). Indeed, there are short trivial protocols with the required input-output behavior, but these are not resilient to malicious parties.

channel, this can be done in n rounds by simply having each party broadcast its bit. In 1988, Gallager [7] showed that $\mathcal{O}(n \log\log n)$ broadcast rounds suffice to solve the problem over the noisy broadcast channel, with polynomially small error probability. Note that this also means that any function on n input bits can be computed over the noisy broadcast channel in $\mathcal{O}(n \log\log n)$ rounds[3], making the identity function "complete" for the computation of such functions. Gallager's result was shown to be tight in the beautiful 2005 paper by Goyal, Kindler, and Saks [10].

The noisy broadcast channel with malicious parties. We consider El Gamal's question in the presence of malicious parties. Specifically, assume that a constant fraction of the parties participating in the protocol are malicious. These parties are controlled by a know-it-all adversary that sees all inputs, as well as all the sent and received bits. The rest of the parties are assumed to be honest and following the protocol, and they may not know the identity of the malicious parties.

Due to the presence of malicious parties, it is unrealistic to expect all parties to compute the identity function (neither over the noiseless broadcast channel, nor over the noisy broadcast channel) for two reasons: (i) the malicious parties will deliberately give incorrect outputs; (ii) a malicious party with an input x^i can behave as if it is holding the input $1 - x^i$, preventing the honest parties from learning its input. Therefore, we relax our requirement and only ask that each *honest* party outputs the input bits of all other *honest* parties. Specifically, we want that for every input x, the following condition is satisfied with high probability:

> $(*)$ *Each party i outputs $(\tilde{x}^i_1, \tilde{x}^i_2, \cdots, \tilde{x}^i_n)$, where if both i and i' are honest, then $\tilde{x}^i_{i'} = x^{i'}$ (otherwise, $\tilde{x}^i_{i'}$ can be arbitrary).*

Over the noiseless broadcast channel, it is easy to see that the simple aforementioned n-round protocol, where each party broadcasts its bit once and outputs all the bits it received, satisfies this relaxed condition. How many rounds of communication are needed when we work over the noisy broadcast channel?

1.1 Our Results

The main result of this paper is a novel $n \cdot \tilde{\mathcal{O}}(\sqrt{\log n})$-round protocol for computing the identity function under the relaxed condition $(*)$, in the presence of a constant fraction of malicious parties and stochastic noise. While more costly than Gallager's protocol, which utterly breaks down in the presence of even a single malicious party (see discussion in Subsection 2.3), our protocol does beat the naive $\mathcal{O}(n \log n)$ protocol[4].

We note that our protocol assumes the *statistical* variant of the noisy broadcast channel (see, e.g., [12, 13]), where the noise flips every sent bit with probability exactly ϵ. Generalizing the protocol for the *fault tolerant* noisy broadcast channel where the noise can flip the j^{th} bit received by the i^{th} party with a different probability for different is and js, as long as these probabilities are all between 0 and ϵ, is left open (see more about this in Subsection 2.6).

[3] To compute the function $g(x^1, x^2, \cdots, x^n)$, the parties run the protocol that computes the identity function. After the protocol, each party knows $x^1, x^2, \cdots, x^n$ and can evaluate $g(\cdot)$ by itself.

[4] In the naive protocol, each party broadcasts its bit $\Theta(\log n)$ times. Party i outputs $(\tilde{x}^i_1, \tilde{x}^i_2, \cdots, \tilde{x}^i_n)$, where $\tilde{x}^i_{i'}$ is the majority of the bits party i received from party i'. If both parties i and i' are honest, the bit $\tilde{x}^i_{i'}$ is the input of party i', except with polynomially small probability, and by a union bound, our relaxed property holds, except with polynomially small probability.

▶ **Theorem 1.** *Let $\epsilon, \theta < 1/2$ and n be large enough. There exists an $n \cdot \tilde{\mathcal{O}}(\sqrt{\log n})$-round randomized protocol with private and public randomness*[5] *over the noisy broadcast channel with noise rate* exactly *ϵ, that computes the identity function in the presence of a θ-fraction of malicious parties (i.e., satisfies condition $(*)$) with error $1/n$. Furthermore, the protocol is computationally efficient – the algorithm for each party runs in almost-linear, i.e., $n^{1+o(1)}$, time.*

As discussed above, due to the "completeness" of the identity function, our result also implies that *any* n-bit function (each party gets a single input bit) can be computed over the noisy broadcast channel in the presence of a constant fraction of malicious parties in $n \cdot \tilde{\mathcal{O}}(\sqrt{\log n})$ rounds.

1.2 Our Techniques and the Notion of Locality Sensitive Codes

The starting point of our construction is Gallager's protocol. At the heart of this protocol is a clever trick that uses (standard) error correcting codes. While Gallager's trick inherently breaks in the presence of malicious parties, we draw inspiration from his ideas and design a different protocol using a new type of codes that we call *locality sensitive codes*.

Roughly speaking, our locality sensitive codes map "close" messages to "close" codewords, while messages that are not close are mapped to codewords that are very far apart. In more detail, our alphabet set is the set of integers. Two messages $m, m' \in \mathbb{Z}^k$ that are close in every coordinate ($|m_i - m'_i| \leq \alpha$ for every $i \in [k]$) will be mapped to codewords that are close in almost every coordinate, but two messages that are far apart in at least one coordinate are mapped to codewords that are far in almost all coordinates. Note that locality sensitive codes are a generalization of classical error correcting codes. Indeed, by setting $\alpha = 0$ we can interpret "closeness" as "equality" (two messages are close only if they are identical) and retrieve the definition of standard error correcting codes – identical messages are mapped to identical codewords and non-identical messages are mapped to codewords with a large distance.

We mention that the definition of locality sensitive codes is reminiscent of that of *locality sensitive hash functions*, often used by algorithms for the *approximate nearest neighbor* problem and other related problems (see, e.g., [2, 3]). However, while locality sensitive hash functions are hash functions, and as such, are contracting the message, locality sensitive codes stretch the message. Since the problem studied in this paper is, at least seemingly, very different from other known applications of locality sensitive hash functions, devising further connections between locality sensitive codes and locality sensitive hashes will be interesting.

1.3 Future Directions

Our work suggests several future directions, we list a few below.

Improving our result. One obvious interesting question is whether our result in Theorem 1 is optimal in terms of communication complexity, or whether it can be improved. In particular, is it possible to match Gallager's construction and design an $\mathcal{O}(n \log \log n)$ protocol for the

[5] Observe that private and public randomness are "incomparable" in our model: the public random string is known to all parties, including the malicious parties. The private random strings are each known to a single party, and, in particular, the private random string of an honest party is not known to any of the malicious parties (see Subsection 3.3).

identity function that handles stochastic noise and also works in the presence of a constant fraction of malicious parties[6]? One direction towards this goal is to improve the construction of locality sensitive error correcting codes.

Interactive codes preserving other properties. In this paper, we consider the property P = "Π is resilient to a constant fraction of malicious parties" and show that in certain cases, protocols Π that satisfy P can be compiled into noise resilient protocols that still respect P. Can this be done for other useful properties P? We mention that work of [9] can be interpreted as asking a similar question with P = "Π is computing some function f privately" (and with adversarial noise), and answering it in the negative.

General interactive coding with malicious parties. As mentioned in Subsection 1.1, Theorem 1 implies a similar simulation for any n-bit function. Thus, one could have hoped that our simulation would also extend to functions with more than n input bits (i.e., the case where parties get long inputs), allowing us to convert any t-round noiseless protocol with malicious parties to a $t \cdot \tilde{\mathcal{O}}(\sqrt{\log n})$-round protocol over the noisy broadcast channel with malicious parties. However, even without the presence of malicious parties, we suspect that the overhead in making a broadcast protocol noise resilient can be a multiplicative factor of $\tilde{\Omega}(\log n)$ (and, in particular, a multiplicative $\mathcal{O}(\log \log n)$ overhead á la Gallager does not always suffice).

Interactive coding with malicious parties over different models. While we believe that there is no scheme with a small overhead that converts protocols in the noiseless broadcast channel that are resilient to malicious parties to protocols in the noisy broadcast channel that are resilient to malicious parties, such a scheme may exist for other channels, such as the (synchronous or asynchronous) peer-to-peer model. If it does, this scheme would imply noise resilient consensus and leader election protocols in the respective models. We mention that noise resilient consensus is considered in [11], under a different noise model.

Better than optimal interactive coding with adversarial noise. Another interesting direction is further exploring the relaxed requirement $(*)$. In this paper, we design a protocol for the identity function that satisfies $(*)$ even in the presence of a constant fraction of malicious parties and stochastic noise. Does such a protocol exist in the presence of a more general type of noise – say, if we allow the adversary to corrupt a different set of parties in each round or if we allow general adversarial noise, where the adversary can corrupt a constant fraction of the received bits? While prior works argue that in multi-party settings it is impossible to handle more than a $1/n$ fraction of adversarial noise, as with this budget, the adversary can corrupt one of the parties completely, this may be possible under a relaxed definition along the lines of $(*)$.

2 Overview of Our Protocol

In this section, we build up to our protocol step by step, covering our main ideas.

2.1 The Identity Problem Over the Broadcast Channel

In the broadcast channel, there are n parties that communicate with one another. This communication happens through bit "broadcasts", namely, bits sent from one of the n parties to *all* the parties. The party sending these bits computes them using the bits it

[6] Goyal et al. [10], proved their lower bounds for the statistical model assumed in this paper, where each received bit is flipped with probability exactly ϵ. Therefore, our result cannot be improved to an $o(n \log \log n)$-round protocol.

received during the communication that happened so far and its own private input. The end-goal of this communication is to compute a joint function of all the private inputs while communicating as few bits as possible.

An important setting (indeed, complete in certain respects) is when all the n parties have a bit as their input, and they want to know the inputs of all the other parties. Formally, party $i \in [n]$ has a bit x^i and should output the bit string $x^1, x^2, \cdots, x^n$ after the communication. We shall call this problem the identity problem in the rest of this document.

The identity problem admits a simple and optimal n round communication protocol over the broadcast channel: In round $i \in [n]$, party i broadcasts their bit x^i to all the parties. After n rounds, all the parties would have received all the bits and can output the string $x^1, x^2, \cdots, x^n$.

This simple protocol boasts of some nice and non-trivial properties. For example, even if an (arbitrarily large) subset of parties do not follow the protocol and are malicious, we still have the guarantee that all non-malicious parties will output the bit of all other non-malicious parties correctly. This property makes sure that, when this protocol is run on a large distributed system, it will be resilient to a subset of the parties failing or being taken over by an adversary. Moreover, the protocol described is also communicationally and computationally efficient.

However, this protocol also has a major weakness in that it crucially relies on the fact that the channel does not corrupt any of the bits sent, which are received exactly by all the parties. Is it possible to have a protocol that is resilient to both malicious parties and channel corruptions?

2.2 The Noisy Broadcast Channel and Gallager's Protocol

To study the above question, one needs to move to the noisy broadcast channel. This channel is identical to the broadcast channel except that it has stochastic noise, namely, there is a parameter $0 < \epsilon < \frac{1}{2}$ such that when any of the parties broadcasts a bit b over the channel, all the parties may either receive b, with (independent) probability $1 - \epsilon$, or may receive the bit $1 - b$, with probability ϵ.

The protocol for the identity problem described above can even be simulated over the noisy broadcast channel, albeit with higher communication. For example, one may repeat each bit broadcast during the protocol $\mathcal{O}(\log n)$ times, and this will ensure that all the parties receive the bit sent except with probability polynomially small in n. A simple union bound over all the n parties and all the n bits shows that the protocol does solve the identity problem except with probability polynomially small in n.

In fact, not only does this protocol solve the identity problem in a way that is resilient to channel corruptions, it also preserves the property of the original protocol of being resilient to malicious parties. Indeed, even if an arbitrarily large subset of parties in the protocol are malicious, all the non-malicious parties will output the bits of all the malicious parties with high probability.

2.2.1 Gallager's Protocol [7]

The main drawback of this protocol is that it communicates $\mathcal{O}(n \log n)$ bits and it is not immediately clear if this is optimal in terms of communication. In fact, without the restriction of being resilient to malicious parties, Gallager, in his elegant work [7], showed that it is provably not so, exhibiting a protocol that communicates $\mathcal{O}(n \log \log n)$ bits and is resilient to channel corruptions[7].

[7] [10] later showed that Gallager's protocol is optimal up to constant factors.

The main idea behind Gallager's improved protocol can be easily understood from an information theoretic viewpoint. Consider the $\mathcal{O}(\log n)$ rounds where a given party broadcasts in the simple $\mathcal{O}(n \log n)$ communication protocol. In the first few rounds when this party broadcasts, each bit broadcast gives a relatively large amount of information about its input to all the other parties. However, as the number of repetitions increases, the other parties already know the input bit with significant probability and each bit sent starts to give lesser and lesser information about the party's input. Thus, by independently repeating their inputs later in the protocol, the parties are wasting a lot of communication to convey a small amount of information about their inputs. This is clearly suboptimal and a more sensible approach is to "combine" the inputs of several of the parties into fewer bits, while maintaining that the information conveyed is the same.

Actually implementing this idea requires ingenuity, and [7] does it by having a protocol with three stages as described below[8]:

- **Stage Broadcast:** This stage of Gallager's protocol has $\mathcal{O}(n \log \log n)$ rounds and in this stage, all the parties broadcast their input $\mathcal{O}(\log \log n)$ times. As the total number of broadcasts per party is small (only $\mathcal{O}(\log \log n)$), all the broadcasts in this stage convey a large amount of information to the other parties.
  Because all the parties broadcast $\mathcal{O}(\log \log n)$ times in stage Broadcast, after this stage, any party can decode (by simple majority based decoding) the input of any other party correctly, except with probability at most $\frac{1}{\text{polylog}(n)}$.
- **Stage Guess:** In this stage, the parties divide themselves into families of size $A = \Theta(\log n)$. As the size of the families is $\Theta(\log n)$, a simple union bound shows that any party in a family can decode the inputs of all other parties in the family correctly, except with probability at most $\frac{1}{\text{polylog}(n)}$.
  In fact, as the noise received by all the parties is independent, we also have concentration and, except with probability polynomially small in n, we have that at least 90% of the parties in a family correctly decode the inputs of all the parties in the family.
- **Stage Boost:** This is the most important stage of the protocol, where the parties "combine" multiple input bits and give information about all of the combined bits in the same communication bit. Recall that before this stage, the parties are divided into families of size A, and, except with probability polynomially small in n, at least 90% of the parties in a family know the input of all the parties in the family.
  We describe stage Boost from the perspective of a single family, noting that the behavior of all the families is symmetric. Party i in this family, for all $i \in [A]$, takes the vector of bits it decoded for all the parties in the family, encodes this vector using a constant rate error correcting code, and broadcasts the i^{th} coordinate of this encoding[9]. Observe that as this coordinate depends on all the decoded bits, transmitting it conveys information about all the A bits in the family. Indeed, it is this combination that reduces the failure probability ("boosts" the success probability) of the protocol from $\frac{1}{\text{polylog}(n)}$ to $\frac{1}{\text{poly}(n)}$ without wasting $\Omega(\log n)$ communication per party and allows us to union bound over all parties and all bits.
  Due to the fact that 90% of the parties in a family encode the same (correct) vector of inputs, at least 90% of the sent coordinates are actually coordinates from one codeword. As the channel corrupts each sent symbol with a small constant probability, for any one of the n parties, it holds, except with probability polynomially small in n, that at least

[8] We note that [7] does not describe his protocol in terms of these stages. However, it will be helpful for us to talk about his protocol in this framework.

[9] We assume in this description that the error correcting code takes a bit string to a string over a larger alphabet but of the same length, and assume for simplicity that the parties can broadcast symbols from this larger alphabet in one round.

80% of the received coordinates come from the same (correct) codeword. Because this is a codeword of a good error correcting code, these 80% of the coordinates suffice to decode the inputs of all parties in the family correctly, except with probability polynomially small in n, and a union bound over all families and all parties finishes the proof of correctness of Gallager's protocol.

2.3 Gallager's Protocol in the Presence of Malicious Parties

Gallager's protocol shows that if resilience to malicious parties is not required, then the simple $\mathcal{O}(n \log n)$ bit communication protocol can be improved. Our main result is stronger, showing that it is possible to do better than the $\mathcal{O}(n \log n)$ bit communication protocol while maintaining its resilience against malicious parties. Before we describe our ideas, however, it will be helpful to first understand where Gallager's protocol fails if some of the parties are malicious.

At first glance, one may mistakenly believe that Gallager's protocol is also resilient to malicious parties. Indeed, the bits broadcast by the parties in stage Broadcast do not depend on the bits received by them, no communication happens in stage Guess, and the argument described in Subsubsection 2.2.1 would work (with slightly different parameters) as long as no family has a lot, say more than 10%, of malicious parties. As the families are size $A = \Theta(\log n)$, this last property can be ensured, except with probability polynomially small in n, by, say, partitioning the parties into families randomly before the execution of the protocol.

However, delving deeper reveals a major problem. Recall that, for stage Boost to work, stage Guess must ensure that, except with probability polynomially small in n, at least 90% of the parties in any family have the same (correct) decoding for the input bits of all the parties in the family. As the parties perform majority based decoding in stage Guess, this is equivalent to saying that, for at least 90% of the parties in a family, the majority bit they receive for all the parties in the family in stage Broadcast is the same (and correct) except with probability polynomially small in n.

The last property is true if there are no malicious parties. Indeed, all parties repeatedly broadcast their input in stage Broadcast, and because the channel only corrupts with a small constant probability, a simple concentration argument shows that, except with probability polynomially small in n, the majority bit received by at least 90% of the parties in a family will be the same as the bit sent.

However, if one of the parties in a family is malicious, it may, in stage Broadcast, broadcast the bit 0 half the time and the bit 1 the other half of the time. This implies that the majority bit received by any other party is equally likely to be 0 or 1, and only around 50% of the parties in the family can agree on the inputs of all the other parties in the family. To make matters worse, if a constant fraction (and not just 1) of the parties in the family are malicious, and all of them behave this way, then no two parties in the family will agree on the inputs of all the parties in the family (with large probability).

Before describing how our protocol gets around this problem, we quickly note that the source of this problem is *not* that the parties perform majority based decoding and decode to a bit only if it is heard at least 50% of the time. Even if the parties have another threshold, say 70%, the malicious parties can simply broadcast 0 in 30% of their broadcasts in stage Broadcast, and 1 in the remaining 70%, and cause the same problem.

2.4 Our Approach: Boosting Before Guessing

We define the "true count" of a party to be the number of times it broadcasts 1 in stage Broadcast. The example from the foregoing section shows that, when there is a malicious party i whose true count is half of the total broadcasts by party i in stage Broadcast, then majority based decoding implies that no more than 50% of the parties in the family of party i agree on the bit they decoded for party i.

Nonetheless, the fact that the channel corrupts only a small constant fraction of the bits sent implies that, except with probability at most $\frac{1}{\text{polylog}(n)}$, all the parties in the family of party i can *approximately* decode the true count for party i. Our main idea is to run stage Boost on these true counts directly, without first executing stage Guess to decode the true counts to bits.

Indeed, if we can run stage Boost on the true counts, and get all the parties to agree on an approximation for the true counts that is correct except with probability polynomially small in n, we can union bound over all parties and all counts to conclude that, except with probability polynomially small in n, all the parties know all the true counts approximately correctly. Once this happens, the parties can discard all counts that are close to $\frac{1}{2}$, as these can only be from malicious parties, and run stage Guess on the remaining counts to get the input bits of all the non-malicious parties correctly.

2.4.1 Computing the Encodings

There is however, a key difference between boosting the true counts and boosting the guessed bits. While boosting the guessed bits, Gallager's protocol ensured that most of the parties in a family have the same Boolean vector $u \in \{0,1\}^A$ of the bits guessed for parties in the family. This allowed an error correcting code of u to be computed in a distributed way, where every party i in the family contributes coordinate i by encoding its Boolean vector.

With the true counts, this is no longer possible as *no* party in the family is likely to have the vector of true counts exactly, and the parties merely possess a good approximation of it. That is, if $p \in \mathbb{Z}^A$ denotes the vector of the true counts for the family, each party in the family now has a different approximation $q \in \mathbb{Z}^A$ of p. The task to be solved now is to compute an encoding of p in a *distributed* manner, when parties only have access to the approximations q!

This task seems to be impossible for standard error correcting codes, for which changing any of the coordinates of the message being encoded even slightly may result in a codeword completely unrelated to the original codeword. What we need instead is an error correcting code that is also "locality sensitive", namely, for two messages (not necessarily Boolean) q_1 and q_2 that are close in all the coordinates, the encodings of q_1 and q_2 are also close in most of the coordinates. On the other hand, if q_1 and q_2 are far in any of the coordinates, then their encodings must be far in almost all of the coordinates.

Connection to locality sensitive hashing. Our description above may remind the reader of the area of locality sensitive hashing, where vectors in $\mathbb{R}^d$ are hashed into buckets such that vectors that are close to each other are likely to be hashed into the same bucket, while vectors that are far apart are likely to be hashed to different buckets.

Our description of locality sensitive error correcting codes is related, but different. Firstly, by definition, a hash function loses information to make the data more manageable while an error correcting code adds more redundant information to make the data resilient to corruptions. Moreover, a locality sensitive hash only requires the hash values of two inputs that are far to be different, while in our definition of locality sensitive error correcting codes,

we will require the codewords to not only be different, but also far apart, in most of the coordinates. In fact, how far they can be will be critical in determining the communication complexity of our protocol.

2.5 Locality Sensitive Error Correcting Codes

As described above, we desire a code C that is locality sensitive. Namely, it has the following properties:

1. **("close" → "close"):** If two messages q_1 and q_2 are such that their coordinate-wise difference is at most α in absolute value, for some $\alpha \geq 0$, then the difference of most of the coordinates of $\mathsf{C}(q_1)$ and the corresponding coordinate of $\mathsf{C}(q_2)$ is at most β, for some $\beta \geq 0$.
2. **("not close" → "far"):** If there exists a coordinate where messages q_1 and q_2 differ by more than α', for some $\alpha' > \alpha$, then the difference of most of the coordinates of $\mathsf{C}(q_1)$ and the corresponding coordinate of $\mathsf{C}(q_2)$ is more than β in absolute value.

2.5.1 Constructing Locality Sensitive Error Correcting Codes

We present a randomized construction of locality sensitive error correcting codes that proceeds in two steps. First, we construct a version of the codes over the alphabet $\mathbb{Z}$ with weak parameters by taking each coordinate of the codeword to be a random linear function of the coordinates of the messages being encoded (details later). The codes we construct in this step will satisfy the property in item 1 above for an extremely large, say 99.99% of the coordinates but will satisfy the property in item 2 for only 50% of the coordinates.

This is followed by an amplification step where we combine independent copies of the codes constructed in the previous step to get better parameters. Specifically, we take L copies, for some constant L, and construct a "joint" codeword each of whose coordinates $\in \mathbb{Z}^L$ is a tuple consisting of the corresponding coordinate in all the codewords. A pair of such joint coordinates is considered to be far if any one of the constituent coordinates is far. By setting L to be the right amount, we can make sure that the constant in item 1 only degrades slightly, say to 99%, while the constant in item 2 improves drastically to 99%.

It remains to describe step 1, the construction of un-amplified locality sensitive error correcting codes. As mentioned earlier, every coordinate of these codes is a random linear function of the coordinates of the message being encoded. The coefficients in this linear function belong to $\{-1, 1\}$ chosen independently and uniformly. A simple concentration argument shows that, for messages q_1 and q_2 of length $k > 0$ whose coordinate-wise difference is at most α, the value of such a random linear function will not differ by more than $\mathcal{O}\left(\alpha\sqrt{k}\right)$, with high probability. Thus, setting $\beta = \mathcal{O}\left(\alpha\sqrt{k}\right)$ ensures that item 1 above is satisfied.

It remains to show why item 2 is satisfied. For this, assume that two messages q_1 and q_2 of length k are being encoded such that q_1 and q_2 differ by at least $\alpha' > \alpha$ in at least one coordinate. Let us assume without loss of generality that this is the last coordinate. Then, for any choice of coefficients for the first $k-1$ coordinates, there exists a choice in $\{-1, 1\}$ of the coefficient for the last coordinate such that the difference in the value of the linear function is at least α' in magnitude. Indeed, either the difference without the last coordinate is positive, in which we can select the value in $\{-1, 1\}$ that will make the difference increase by α', or it is negative, in which case we can select the other value and make the difference decrease by α'. In either case, the resulting difference is at least α' in absolute value.

This implies that, with probability at least $\frac{1}{2}$, setting $\beta = \alpha'$ satisfies item 2 finishing the argument.

2.6 Our Protocol

Armed with locality sensitive error correcting codes, we now describe our protocol. Note that our description has parameters such as k, m, α, etc. and we set these parameters to appropriate values later in the sketch. Recall that the n parties are divided into families of size $A = \Theta(\log n)$ randomly and this ensures that, except with probability polynomially small in n, all families have a small fraction of malicious parties.

We divide our protocol into the same three stages as in Gallager's protocol, although the order of the stages is different.

- **Stage Broadcast:** In this stage, all the parties broadcast their input m times (we set m later in this sketch). The parameter m will be chosen so that, except with probability at most $\frac{1}{\text{polylog}(n)}$, all the parties in a family receive approximately the same counts of the number of 1s from all other parties in the family up to an additive error of α.
  As in Gallager's protocol, as the noise received by each party is independent, except with probability polynomially small in n, at least 90% of the parties in any family receive the same counts up to an additive error of α.
- **Stage Boost:** Notwithstanding the high level similarity to it, in that the parties in a family join forces to convince all the parties of the approximately correct counts, stage Boost in our protocol is very different from stage Boost in Gallager's protocol.
  In our protocol, we first divide the families into $\frac{A}{k}$ groups of k parties each. Then, for all i, party i in the family computes, using a locality sensitive error correcting code, an encoding for each group in the family of the vector of counts it received from that group in stage Broadcast, and broadcasts coordinate i of all the $\frac{A}{k}$ encodings.
  All the parties then combine the coordinates received from the different parties in a family to get a codeword for all the $\frac{A}{k}$ groups in the family, and decode it to get the individual counts for the parties in the group. We note that these decoded counts are within α' of the true counts. Indeed, 90% of the parties in the family sent coordinates from encodings of counts that were within α of the true counts. Only a small constant fraction of coordinates were altered due to corruptions, and another small number of coordinates were sent by malicious parties in each family. Therefore, most of the coordinates received are from encodings of vectors within α of the true counts. Due to the property in item 1 above, these coordinates are within β of the corresponding coordinates in the encoding of the true counts.
  Because of the large number of coordinates that are within β of the corresponding coordinates in the encoding of the true counts, due to item 2 above, these coordinates are unlikely to come from a vector that is not within α' of the true counts. Consequently, except with probability polynomially small in n, all the parties decode to a vector of counts within α' of the true counts, as desired. We will set α' to be the same order of magnitude as m, but smaller, say $\alpha' = \frac{m}{10}$.
- **Stage Guess:** In this stage, the parties simply perform majority based decoding of the counts decoded after stage Boost. This results in correct outputs for non-malicious parties because their true counts are either 0, if they have input 0, or m, if they have input 1. As the decoded counts are only $\alpha' = \frac{m}{10}$ off from the true counts, majority based decoding will work correctly for the non-malicious parties.

Setting the parameters. Recall the size of a family is $A = \Theta(\log n)$. We already explained $\alpha' = \frac{m}{10}$, and our construction of locality sensitive error correcting codes in Subsection 2.5 implies $\beta = \mathcal{O}\left(\alpha\sqrt{k}\right) = \frac{m}{10}$ as well. The number of bits communicated by a given party in our protocol is m in stage Broadcast and (roughly) $\frac{A}{k}$ in stage Boost. Thus, total communication by a party equals

$$m + \frac{A}{k} = m + \mathcal{O}\left(A \cdot \frac{\alpha^2}{m^2}\right).$$

In the fault tolerant model, where the best guarantee obtainable is $\alpha = \Theta(m)$, the expression above is at least $\Omega(\log n)$ and we do not get any gains over the protocol in Subsection 2.2. However, in the statistical model, where the more predictable nature of the noise allows us to use concentration bounds and get $\alpha = \Theta(\sqrt{m})$, the expression above is minimized when $m = \Theta\left(\sqrt{\log n}\right)$, and we get an improvement over the protocol in Subsection 2.2. Observe that, perhaps surprisingly, the malicious parties are weaker in the model with "more" noise, and this is because the noise is also more predictable.

Mixability of the encodings. We finish the section by covering one subtlety about stage Boost that we omitted from the description above. In our implementation, instead of party i in the family sending coordinate i of the encoding it computed, we have it send a random coordinate along with the identity of the coordinate. As a typical coordinate will satisfy the properties in item 1 and item 2 above, this does not affect our analysis by much, but it does help us avoid the pathological case where all parties send a coordinate that is atypical in that it does not satisfy item 1 and item 2.

We note that this pathological case never arose in Gallager's implementation as there, the coordinates sent by different parties in a family were different coordinates from the encodings of the *same* message, and thus, only a small number of them could possibly be atypical. For us, the parties send encodings of *nearby* messages, and it is possible that all coordinates that are sent are atypical for the encoding where they came from, hence, this extra randomization step.

3 Models and Formal Problem Definition

3.1 Noisy Copies

Let $\epsilon \in [0, 1/2)$ be a noise parameter and $k \in \mathbb{N}$. An *ϵ-noise bit* is a $\{0,1\}$-valued random variable that takes value 1 with probability *exactly* ϵ. An *ϵ-noise k-vector* N is a sequence of k independent ϵ-noise bits. For a bit-vector $x \in \{0,1\}^k$, an *ϵ-noisy copy of x* is a random variable of the form $x \oplus N$, where $\oplus$ denotes the bitwise XOR, and N is an ϵ-noise k-vector. More generally, if X is any random variable taking values in $\{0,1\}^k$ an *ϵ-noisy copy of X* is a random variable of the form $X \oplus N$, where N is an ϵ-noise k-vector chosen independently from X.

3.2 The Noisy Broadcast Model

The (statistical) noisy broadcast model considers n parties $P_1, \cdots, P_n$. Let $\mathcal{X}$ and $\mathcal{Y}$ be sets. The input is a vector $x \in \mathcal{X}^n$, and each of the parties P_i initially has coordinate $x^i \in \mathcal{X}$. The goal is for party P_i to evaluate a function $f^i : \mathcal{X}^n \to \mathcal{Y}$ at x, i.e., output $y^i = f^i(x)$. This goal is to be accomplished by a noisy broadcast protocol.

The specification of a (deterministic) protocol over the noisy broadcast model with noise rate *exactly* ϵ consists of:

1. The number $s \in \{0\} \cup \mathbb{N}$ of broadcasts used in the protocol.
2. A sequence $i_1, \cdots, i_s \in [n]$ of indices of parties (with repetitions allowed).
3. A sequence $g_1, \cdots, g_s$ of broadcast functions, where $g_j : \mathcal{X} \times \{0,1\}^{j-1} \to \{0,1\}$.
4. For all i, an output function $h^i : \mathcal{X} \times \{0,1\}^s \to \mathcal{Y}$ for P_i.

Running a protocol. The execution of a noisy broadcast protocol Π depends on the input x, and on a noise vector N. We will think of N as a concatenation of s independent ϵ-noise n-vectors $N_1, \cdots, N_s$. Let $j \in [s]$. In the j^{th} step of the execution of Π, party P_{i_j} broadcasts a bit b_j and all the parties $i \in [n]$, receive $b_j^i = b_j \oplus N_j[i]$, an independent noisy copy of b_j. Formally, the bit broadcast by P_{i_j} at step j is $b_j = g_j(x^{i_j}, b_1^{i_j}, b_2^{i_j}, \cdots, b_{j-1}^{i_j})$, that is, the value of g_j on the input of P_{i_j} and the $j-1$ bits received by P_{i_j} during the first $j-1$ rounds. The output of P_i is $y^i = h^i(x^i, b_1^i, \cdots, b_s^i)$, that is, the value of h^i on the input of P_i and the s-vector of bits received by P_i.

Note that the assumed model is *non-adaptive* or *oblivious*: the sequence of parties who broadcast is fixed in advance and does not depend on the execution. Without this requirement, the noise could lead to several parties speaking at the same time (a *collision*). Moreover, this problem will be more serious when we introduce malicious parties, who may decide to speak all the time causing collisions in every round. Finally, note that the model rules out *communication by silence*: when it is the turn of a party to speak, it must speak.

3.3 The Noisy Broadcast Model with Malicious Parties

So far, we assumed that all participating parties are collaborating and following the protocol. We now consider the model where a subset $\mathsf{Mal} \subset [n]$ of the parties, called the *malicious parties*, are controlled by an adversary and may not follow the protocol. The set of malicious parties is determined prior to the execution of the protocol and is unchanged throughout the duration of the execution. We consider randomized protocols in this malicious setting and allow parties to use both private randomness (known to a single party) and public randomness (known to all parties). Observe that in the standard (non-malicious) setting, public randomness can always be used in lieu of private randomness. This is no longer possible in our malicious setting, as will be apparent next.

The adversary controlling the malicious parties is assumed to know the inputs of all the parties, the shared random string, and the private random strings of the parties in Mal. In addition, in round $j \in [s]$, the adversary knows the channel's prior noise vectors $N_1, N_2, \cdots, N_{j-1}$, and thus also knows all the bits that were sent and received by all the parties in all the previous rounds. (Note that the adversary does not know either the private random strings of the honest parties or the noise in the channel in future rounds). The parties in $[n] \setminus \mathsf{Mal}$ are still assumed to be following the protocol and are called *honest parties*.

Computing functions. Let $x \in \mathcal{X}^n$ be an input for the parties and let Mal be the set of malicious parties. We say that the input $x' \in \mathcal{X}^n$ is *consistent* with x and Mal if $\hat{x}^i = x^i$ for every $i \in [n] \setminus \mathsf{Mal}$.

Let $\theta \in [0, 1/2)$ and assume that n is a sufficiently large function of θ. Let $\delta \geq 0$. We say that a randomized protocol Π over the noisy broadcast channel with noise rate exactly ϵ *computes the functions* $\{f^i\}_{i\in[n]}$ *in the presence of* θ*-fraction of malicious parties with error* δ if whenever the set of malicious parties Mal satisfies $|\mathsf{Mal}| \leq \theta n$, then for every $x \in \mathcal{X}^n$, with probability at least $1 - \delta$, the following holds: For every $i \in [n] \setminus \mathsf{Mal}$, there exists $\hat{x} \in \mathcal{X}^n$ that is consistent with x and Mal, such that the output of P_i in Π is $y^i = f^i(\hat{x})$. Here, the probability is over the private and public randomness and the noise in the channel.

We say that a randomized protocol Π over the noisy broadcast channel with noise rate exactly ϵ *computes the identity function in the presence of* θ*-fraction of malicious parties with error* δ if $\mathcal{X} = \{0,1\}$ and $\mathcal{Y} = \{0,1\}^n$ and Π computes the functions $\{f^i\}_{i\in[n]}$ in the presence of θ-fraction of malicious parties with error δ, where $f^i : \{0,1\}^n \to \{0,1\}^n$ is given by $f^i(x) = x$. Simplified for the identity function, the above means that whenever $|\mathsf{Mal}| \leq \theta n$,

then, for every input x, with probability at least $1-\delta$, the following holds: For all $i \in [n]$, P_i outputs a vectors $y^i = (\tilde{x}_1^i, \tilde{x}_2^i, \cdots, \tilde{x}_n^i)$, where if $i, i' \in [n] \setminus \mathsf{Mal}$, then $\tilde{x}_{i'}^i = x^{i'}$ (otherwise, $\tilde{x}_{i'}^i$ can be arbitrary), as suggested by $(*)$.

4 Preliminaries

Please refer to the full version for the missing proofs.

4.1 Concentration Inequalities

▶ **Lemma 2** (Multiplicative Chernoff bound). *Suppose $X_1, \cdots, X_n$ are independent random variables taking values in $[0,1]$. Let X denote their sum and let $\mu = \mathbb{E}[X]$ denote the sum's expected value. Then,*

$$\Pr\left(X \geq (1+\delta)\mu\right) \leq \mathrm{e}^{-\frac{\delta^2\mu}{2+\delta}}, \qquad \forall 0 \leq \delta,$$

$$\Pr\left(X \leq (1-\delta)\mu\right) \leq \mathrm{e}^{-\frac{\delta^2\mu}{2}}, \qquad \forall 0 \leq \delta \leq 1.$$

In particular, we have that:

$$\Pr\left(X \geq (1+\delta)\mu\right) \leq \mathrm{e}^{-\frac{\delta\mu}{3}\cdot\min(\delta,1)}, \qquad \forall 0 \leq \delta,$$

$$\Pr\left(|X-\mu| \geq \delta\mu\right) \leq 2\cdot\mathrm{e}^{-\frac{\delta^2\mu}{3}}, \qquad \forall 0 \leq \delta \leq 1.$$

We derive a couple of corollaries of Lemma 2, that will be convenient for us to use.

▶ **Corollary 3.** *Suppose $X_1, \cdots, X_n$ are independent random variables taking values in $[0,1]$. Let X denote their sum and let $\mu = \mathbb{E}[X]$ denote the sum's expected value. Then, for all $\Delta \geq 2\mu$, we have:*

$$\Pr(X \geq \Delta) \leq \mathrm{e}^{-\frac{\Delta}{6}}.$$

Proof. As $\Delta \geq 2\mu$, we have $\Delta = (1+\delta)\mu$ for some $\delta \geq 1$. Applying Lemma 2 with this δ, we get $\Pr(X \geq \Delta) \leq \mathrm{e}^{-\frac{\Delta-\mu}{3}} \leq \mathrm{e}^{-\frac{\Delta}{6}}$, as desired. ◀

▶ **Corollary 4.** *Suppose $X_1, \cdots, X_n$ are independent random variables taking values in $[-1,1]$ such that $\mathbb{E}[X_i] = 0$ for all $i \in [n]$. If $X = \sum_{i=1}^n X_i$ denotes their sum, then, for all $0 \leq \delta \leq 1$, we have*

$$\Pr(|X| \geq \delta n) \leq 2\cdot\mathrm{e}^{-\frac{\delta^2 n}{6}}.$$

Proof. Apply Lemma 2 on the variables $\frac{X_i+1}{2}$. ◀

We shall also use the following version of Chernoff bound for negatively correlated random variables:

▶ **Definition 5** (Negatively Correlated Random Variables). *For $n > 0$, let $X_1, \cdots, X_n$ be random variables that take values in $\{0,1\}$. We say that the random variables $X_1, \cdots, X_n$ are negatively correlated if for all subsets $S \subseteq [n]$, we have $\Pr(\forall i \in S : X_i = 1) \leq \prod_{i\in S}\Pr(X_i = 1)$.*

▶ **Lemma 6** (Generalized Chernoff Bound; cf. [14]). *For $n > 0$, let $X_1, \cdots, X_n$ be negatively correlated random variables that take values in $\{0,1\}$. Let X denote their sum and let $\mu = \mathbb{E}[X]$ denote the sum's expected value. Then, for any $\delta \geq 0$, we have:*

$$\Pr\left(X > (1+\delta)\cdot\mu\right) \leq \mathrm{e}^{-\frac{\delta^2\mu}{2+\delta}}.$$

In particular, we have that:

$$\Pr\left(X > (1+\delta)\mu\right) \leq \mathrm{e}^{-\frac{\delta\mu}{3}\cdot\min(\delta,1)}, \qquad \forall \delta \geq 0.$$

4.2 Results From Coding Theory

We use the following standard result for error-correcting codes, and include a proof for completeness.

▶ **Lemma 7.** *Let $\delta > 0$ and define $K_0 = \lceil 10/\delta^2 \rceil$. For all $n > 0$, there exists a function $\mathsf{ECC}_{n,\delta} : \{0,1\}^n \to \{0,1\}^{K_0 n}$ such that for all $s \neq t \in \{0,1\}^n$, we have*

$$\Delta(\mathsf{ECC}_{n,\delta}(s), \mathsf{ECC}_{n,\delta}(t)) > \left(\frac{1}{2} - \delta\right) \cdot K_0 n.$$

5 Locality Sensitive Error Correcting Codes

Let $n > 0$ and $x, y \in \mathbb{Z}^n$ be vectors. For $z \in \mathbb{Z}$, define

$$\mathsf{sep}_z(x, y) = \{i \in [n] \mid |x_i - y_i| > z\}.$$

Observe that $|\mathsf{sep}_0(\cdot)|$ is simply the Hamming distance between the vectors x and y, and is therefore, a metric. For general $z \in \mathbb{Z}$ however, the function $|\mathsf{sep}_z(\cdot)|$ may not be a metric. In this paper, we work with the following generalization of the function $\mathsf{sep}(\cdot)$.

▶ **Definition 8.** *Let $n, L > 0$ and $x, y \in \left(\mathbb{Z}^L\right)^n$ be vectors. For $z \in \mathbb{Z}$, define*

$$\mathsf{sep}_z(x, y) = \{i \in [n] \mid \exists l \in [L] : |x_{i,l} - y_{i,l}| > z\}.$$

▶ **Lemma 9.** *Let $n, L > 0$ and $x_1, x_2, y \in \left(\mathbb{Z}^L\right)^n$ be vectors. For $z_1, z_2 \in \mathbb{Z}$, we have*

$$\mathsf{sep}_{z_1}(x_1, y) = \emptyset \wedge \mathsf{sep}_{z_1+z_2}(x_2, y) \neq \emptyset \implies \mathsf{sep}_{z_2}(x_1, x_2) \neq \emptyset.$$

5.1 Definition

We now define locality sensitive error correcting codes. Throughout this section and the next, we fix integers $m, k > 100$.

▶ **Definition 10.** *Let $n, L > 0$ and $\mathsf{C} : (\{0\} \cup [m])^k \to \left(\mathbb{Z}^L\right)^n$. Let $\alpha, \alpha', \beta, \mu, \mu' > 0$ be parameters. We say that the function C is an $(n, L, \alpha, \alpha', \beta, \mu, \mu')$-locality sensitive error correcting code if it has the following two properties:*

- **(α, β, μ)-locality sensitive:** *For all $x, y \in (\{0\} \cup [m])^k$, we have:*

  $$\mathsf{sep}_\alpha(x, y) = \emptyset \implies \left|\mathsf{sep}_\beta(\mathsf{C}(x), \mathsf{C}(y))\right| \leq (1 - \mu)n.$$

- **(α', β, μ')-error correcting:** *For all $x, y \in (\{0\} \cup [m])^k$, we have:*

  $$\mathsf{sep}_{\alpha'}(x, y) \neq \emptyset \implies \left|\mathsf{sep}_\beta(\mathsf{C}(x), \mathsf{C}(y))\right| \geq \mu' n.$$

We sometimes say C is a locality sensitive error correcting code or C is an (n, L)-locality sensitive error correcting code if we do not wish to emphasize the other parameters. Our protocol requires C to have a short representation as stated below:

▶ **Definition 11** (Representation Length)**.** *Let $n, L > 0$ and C be an (n, L)-locality sensitive error correcting code. Define:*

$$\|\mathsf{C}\| = \max_{x \in (\{0\} \cup [m])^k} \max_{i \in [n]} \max_{l \in [L]} |\mathsf{C}_{i,l}(x)|.$$

Observe that for all $x \in (\{0\} \cup [m])^k$, the value of $\mathsf{C}(x)$, can be encoded using $10 \cdot \log(10 \cdot \|\mathsf{C}\| + 10)$ bits. We shall use this to show that our protocol does not communicate a lot of bits.

5.2 Proof of Existence

We have:

▶ **Theorem 12.** *Let $\theta < \frac{1}{2}$ and $n \geq \left(\frac{100}{1-2\theta}\right)^3 \cdot k \log m$. There exists a locality sensitive error correcting code C with parameters:*

$$\left(n, 10 \cdot \log \frac{10}{1-2\theta}, 4\sqrt{m \log k}, \beta, \beta, \frac{1}{2}+\theta, \frac{1}{2}+\theta\right),$$

where $\beta = 24\sqrt{mk \cdot \log k \cdot \log \frac{10}{1-2\theta}}$. Moreover, C satisfies $\|\mathsf{C}\| \leq mk$ and C can be computed in time polynomial in (m, k, n, L)[10]

6 Mixability of Codewords

We show that the encodings of similar (but not identical) inputs under a locality sensitive error correcting code can be "mixed" together while maintaining properties similar to the original codewords. In order to formalize this, we first generalize the function $\mathsf{sep}(\cdot)$ from Definition 8.

▶ **Definition 13.** *Let $n, L > 0$, $x \in \left(\mathbb{Z}^L\right)^n$, and $z \in \mathbb{Z}$. For $n' > 0$ and a vector of pairs $y \in \left([n] \times \left(\mathbb{Z}^L\right)\right)^{n'}$, define*

$$\mathsf{sep\text{-}mix}_z(x, y) = \left\{i \in [n'] \mid \exists l \in [L] : \left|x_{y_{i,1},l} - (y_{i,2})_l\right| > z\right\}.$$

Intuitively, every coordinate in y has two components, the first points to a coordinate in x and the second one is a value in $\mathbb{Z}^L$ (a symbol in the code's alphabet). The function $\mathsf{sep\text{-}mix}_z(\cdot)$ compares each coordinate in y to the coordinate it points to in x and checks if they "differ" by more than z.

An easy corollary of the above definition is the following where for a set $S \subseteq [n']$, we use the notation $y|_S$ to denote the vector y restricted to the coordinates in S.

▶ **Corollary 14.** *Let $n, L > 0$, $x \in \left(\mathbb{Z}^L\right)^n$, and $z \in \mathbb{Z}$. Also, let $n' > 0$ and $y \in \left([n] \times \left(\mathbb{Z}^L\right)\right)^{n'}$. We have for all $S \subseteq [n']$ that:*

$$\mathsf{sep\text{-}mix}_z(x, y) \cap S = \mathsf{sep\text{-}mix}_z(x, y|_S).$$

Fix an $(n, L, \alpha, \alpha', \beta, \mu, \mu')$-locality sensitive error correcting code C for the rest of this section. We show that:

▶ **Lemma 15.** *Let $x \in (\{0\} \cup [m])^k$ and $n' > 0$. For $i' \in [n']$, let $y_{i'} \in (\{0\} \cup [m])^k$ be given. If, for $j_1, j_2, \cdots, j_{n'} \in [n]$, we have*

$$Y(j_1, j_2, \cdots, j_{n'}) = \left((j_1, \mathsf{C}_{j_1}(y_1)), (j_2, \mathsf{C}_{j_2}(y_2)), \cdots, (j_{n'}, \mathsf{C}_{j_{n'}}(y_{n'}))\right),$$

then, when $j_1, j_2, \cdots, j_{n'}$ are sampled uniformly at random, we have for all $\Delta_1 \geq 2 \cdot \left(n' - \frac{n'}{n} \cdot \min_{i' \in [n']} \left|\mathsf{sep}_\beta(\mathsf{C}(x), \mathsf{C}(y_{i'}))\right|\right)$ that:

$$\Pr\left(n' - \left|\mathsf{sep\text{-}mix}_\beta(\mathsf{C}(x), Y(j_1, j_2, \cdots, j_{n'}))\right| \geq \Delta_1\right) \leq \exp\left(-\frac{\Delta_1}{6}\right).$$

We also have, for all $\Delta_2 \geq 2 \cdot \frac{n'}{n} \cdot \max_{i' \in [n']} \left|\mathsf{sep}_\beta(\mathsf{C}(x), \mathsf{C}(y_{i'}))\right|$ that

$$\Pr\left(\left|\mathsf{sep\text{-}mix}_\beta(\mathsf{C}(x), Y(j_1, j_2, \cdots, j_{n'}))\right| \geq \Delta_2\right) \leq \exp\left(-\frac{\Delta_2}{6}\right).$$

[10] In fact, as C will only encode messages of logarithmic length in our protocol, the running time of our protocol will be almost-linear even if encoding C took sub-exponential time.

7 Our Protocol

We are now ready to state our protocol. Recall that there are n parties amongst which at most θn are malicious. The following hold:

$$\theta < \frac{1}{2}, \qquad \epsilon = \frac{1}{10}, \qquad n > 100^{100^{\frac{100}{1-2\theta}}}. \tag{1}$$

We define the following parameters:

$$\begin{aligned} k &= \frac{\sqrt{\log n}}{\left(\frac{1}{2}-\theta\right)^2} \cdot \frac{1}{\left(\log \frac{10}{1-2\theta}\right)^2}, & m &= 50000k \cdot \left(\log \frac{10}{1-2\theta}\right)^2 \cdot \log\log n, \\ \alpha &= 4\sqrt{m \log k}, & \beta &= 24\sqrt{mk \cdot \log k \cdot \log \frac{10}{1-2\theta}} < \frac{m}{6}, \\ L &= 10 \cdot \log \frac{10}{1-2\theta}, & A &= 10^{20} \cdot \frac{\log n}{\left(\frac{1}{2}-\theta\right)^3} \cdot \frac{4\theta}{\min(4\theta, 1-2\theta)}, \\ \ell &= 10^{10} \cdot \log\left(\frac{\log n}{\frac{1}{2}-\theta}\right). \end{aligned} \tag{2}$$

For the rest of this paper, we reserve $\mathsf{C} : (\{0\} \cup [m])^k \to \left(\mathbb{Z}^L\right)^A$ to be a locality sensitive error correcting code with parameters:

$$\left(A, L, \alpha, \beta, \beta, \frac{199+2\theta}{200}, \frac{199+2\theta}{200}\right).$$

Such a code is promised by Theorem 12. We have by Definition 10 that, for all $x, y \in [m]^k$,

$$\begin{aligned} \mathsf{sep}_\alpha(x,y) = \emptyset &\implies \left|\mathsf{sep}_\beta(\mathsf{C}(x), \mathsf{C}(y))\right| \le A \cdot \left(\frac{1-2\theta}{200}\right). \\ \mathsf{sep}_\beta(x,y) \neq \emptyset &\implies \left|\mathsf{sep}_\beta(\mathsf{C}(x), \mathsf{C}(y))\right| \ge A \cdot \left(\frac{199+2\theta}{200}\right). \end{aligned} \tag{3}$$

Additionally, we reserve ECC to be the one promised by Lemma 7 for $n \leftarrow \ell$ and $\delta \leftarrow \frac{1}{10}$. By Lemma 7, we have $s \neq t \in \{0,1\}^\ell$ that:

$$\Delta(\mathsf{ECC}(s), \mathsf{ECC}(t)) > \frac{2}{5} \cdot 1000\ell = 400\ell. \tag{4}$$

7.1 Partitioning the Parties

The n parties are randomly divided into n/A families of A parties each. Each family is further divided into A/k groups of k parties each. Formally, this division is performed using the following random process: First, sample a permutation uniformly at random from the set S_n of permutations on $[n]$. Then, for $a \in [n/A]$, family a is the set of parties

$$F_a = \{\sigma(A(a-1)+1), \sigma(A(a-1)+2), \cdots, \sigma(A(a-1)+A)\}.$$

For $b \in [A/k]$, group b in family F_a is the set of parties

$$G_{a,b} = \{\sigma(A(a-1)+k(b-1)+1), \cdots, \sigma(A(a-1)+k(b-1)+k)\}$$

As the families are chosen randomly, no family has a lot of malicious parties with high probability. Formally,

▶ **Lemma 16.** *It holds that:*

$$\Pr_{\sigma \sim \mathsf{S}_n}\left(\exists a \in [n/A] : |F_a \cap \mathsf{Mal}| \ge A \cdot \frac{1+2\theta}{4}\right) \le \frac{1}{n^{30}}.$$

7.2 Our Protocol

For the rest of this section and the analysis, fix a partition of the parties into families such that no family has more that $\frac{1+2\theta}{4}$ fraction of malicious parties. Due to Lemma 16, this happens except with probability at most $\frac{1}{n^{30}}$. We note that this partition allows us to denote a party $i \in [n]$ using either $(a, j) \in [n/A] \times [A]$, emphasizing its family and index within the family, or $(a, b, c) \in [n/A] \times [A/k] \times [k]$, emphasizing its family, its group, and the index within the groups. These ways to denote a party i are equivalent and we use them interchangeably.

Our protocol is symmetric for all the parties and is described in Algorithm 1 from the perspective of party i. In the algorithm, we sometimes use "partial indexing", e.g., we write $q_{a,b'}$ to mean the concatenation of the values $q_{a,b',c'}$ for all possible values of c'.

Please refer to the full version for details about the analysis.

Algorithm 1 Our protocol from the perspective of party $i = (a, j) = (a, b, c)$.

Stage Broadcast:

1: Broadcast input x^i a total of m times.
2: For $b' \in [A/k]$ and $c' \in [k]$, let $q_{a,b',c'} \leftarrow$ number of 1s received from party (a, b', c').

Stage Boost:

3: For $b' \in [A/k]$, sample $z_{a,b'}$ privately and uniformly from $[A]$. Let $v_{a,b'} \leftarrow \mathsf{C}_{z_{a,b'}}(q_{a,b'})$. Broadcast $\mathsf{ECC}((z_{a,b'}, v_{a,b'}))$ (note that ℓ bits suffice to encode $(z_{a,b'}, v_{a,b'})$).
4: For $a' \in [n/A]$, $j' \in [A]$, and $b'' \in [A/k]$, decode the b''^{th} value received from party (a', j') to get $\tilde{\mathsf{C}}_{a',b'',j'} = (\tilde{z}_{a',b'',j'}, \tilde{v}_{a',b'',j'})$.

Stage Guess:

5: For $a' \in [n/A]$ and $b' \in [A/k]$, $\tilde{p}_{a',b'} \leftarrow \arg\min_{p' \in (\{0\} \cup [m])^k} \left|\mathsf{sep\text{-}mix}_\beta(\mathsf{C}(p'), \tilde{\mathsf{C}}_{a',b'})\right|$.
6: For $i' = (a', b', c') \in [n]$, output $\tilde{x}_{i'} = \mathbb{1}\left(\tilde{p}_{a',b',c'} \geq \frac{m}{2}\right)$.

References

1 Noga Alon, Mark Braverman, Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Reliable communication over highly connected noisy networks. In *Symposium on Principles of Distributed Computing (DISC)*, pages 165–173. ACM, 2016.

2 Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008. doi:10.1145/1327452.1327494.

3 Alexandr Andoni, Piotr Indyk, and Ilya P. Razenshteyn. Approximate nearest neighbor search in high dimensions. *CoRR*, abs/1806.09823, 2018. arXiv:1806.09823.

4 Mark Braverman, Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Constant-rate coding for multiparty interactive communication is impossible. In *Symposium on Theory of Computing (STOC)*, pages 999–1010. ACM, 2016.

5 Klim Efremenko, Gillat Kol, and Raghuvansh Saxena. Interactive coding over the noisy broadcast channel. In *Symposium on Theory of Computing (STOC)*, pages 507–520. ACM, 2018.

6 Uriel Feige and Joe Kilian. Finding OR in a noisy broadcast network. *Information Processing Letters*, 73(1-2):69–75, 2000.

7 Robert G. Gallager. Finding parity in a simple broadcast network. *IEEE Transactions on Information Theory*, 34(2):176–180, 1988.

8 Abbas El Gamal. Open problems presented at the 1984 workshop on specific problems in communication and computation sponsored by bell communication research. *"Open Problems in Communication and Computation", by Thomas M. Cover and B. Gopinath (editors). Springer-Verlag*, 1987.

9 Ran Gelles, Amit Sahai, and Akshay Wadia. Private interactive communication across an adversarial channel. *IEEE Trans. Inf. Theory*, 61(12):6860–6875, 2015. doi:10.1109/TIT.2015.2483323.

10 Navin Goyal, Guy Kindler, and Michael Saks. Lower bounds for the noisy broadcast problem. *SIAM Journal on Computing*, 37(6):1806–1841, 2008.

11 Kokouvi Hounkanli, Avery Miller, and Andrzej Pelc. Global synchronization and consensus using beeps in a fault-prone multiple access channel. *Theoretical Computer Science*, 806:567–576, 2020.

12 Eyal Kushilevitz and Yishay Mansour. An $\omega(d \log(n/d))$ lower bound for broadcast in radio networks. *SIAM J. Comput.*, 27(3):702–712, 1998.

13 Ilan Newman. Computing in fault tolerance broadcast networks. In *Computational Complexity Conference (CCC)*, pages 113–122, 2004.

14 Alessandro Panconesi and Aravind Srinivasan. Randomized distributed edge coloring via an extension of the chernoff-hoeffding bounds. *SIAM J. Comput.*, 26(2):350–368, 1997. doi:10.1137/S0097539793250767.

15 Sridhar Rajagopalan and Leonard J. Schulman. A coding theorem for distributed computation. In *Symposium on the Theory of Computing (STOC)*, pages 790–799, 1994.

16 Leonard J Schulman. Communication on noisy channels: A coding theorem for computation. In *Foundations of Computer Science (FOCS)*, pages 724–733. IEEE, 1992.

17 Leonard J Schulman. Deterministic coding for interactive communication. In *Symposium on Theory of computing (STOC)*, pages 747–756. ACM, 1993.

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