Multiparty Interactive Communication with Private Channels

Abhinav Aggarwal
University of New Mexico
Albuquerque, NM
abhiag@unm.edu

Varsha Dani
University of New Mexico
Albuquerque, NM
varsha@cs.unm.edu

Thomas P. Hayes
University of New Mexico
Albuquerque, NM
hayes@cs.unm.edu

Jared Saia
University of New Mexico
Albuquerque, NM
saia@cs.unm.edu

***Abstract*—A group of n players wants to run a distributed protocol $\mathcal{P}$ over a network where communication occurs via private point-to-point channels. Unfortunately, an adversary, who knows $\mathcal{P}$, is able to maliciously flip bits on the channels. Can we efficiently simulate $\mathcal{P}$ in the presence of such an adversary?**

We show that this is possible, even when L, the number of bits sent in $\mathcal{P}$, and T, the number of bits flipped by the adversary are not known in advance. In particular, we show how to create a robust version of $\mathcal{P}$, $\mathcal{P}'$ such that 1) $\mathcal{P}'$ fails with probability at most δ, for any $\delta > 0$; and 2) $\mathcal{P}'$ sends $\tilde{O}(L+T)$ bits, where the $\tilde{O}$ notation hides a $\log(nL/\delta)$ term multiplying L. Critically, our result requires that $\mathcal{P}$ be a protocol that runs correctly in an asynchronous network; in contrast, our protocol $\mathcal{P}'$ must run in a synchronous network.

Additionally, we show how to improve this result when the average message size is not constant. In particular, we show how to create a protocol $\mathcal{P}'$ that sends $O(L(1+(1/\alpha)\log(nL/\delta))+T)$ bits, where α is the average message size. This new $\mathcal{P}'$ is adaptive in that it does not require *a priori* knowledge of α. We note that if α is $\Omega(\log(nL/\delta))$, then this improved algorithm sends only $O(L+T)$ bits, and is therefore within a constant factor of optimal.

***Index Terms*—Interactive Communication, Private channels, Key Exchange**

I. Introduction

How can we compute in the presence of noise? The problem of interactive communication formalizes this problem [40], [37], [1]. In this problem, n players are in a network connected by point-to-point binary symmetric channels, and we want to simulate a distributed protocol $\mathcal{P}$, even when an adversary can flip bits on any of the channels at any time. Our goal is to create a new protocol $\mathcal{P}'$ that correctly simulates $\mathcal{P}$ even in the presence of such adversarial bit flipping.

Under certain special conditions on $\mathcal{P}$, it is already known how to create a $\mathcal{P}'$ with a multiplicative blowup in the number of bits sent that is $O(\log n)$ [37], or even $O(1)$ for certain network topologies [1], [29], [28]. However, the conditions on $\mathcal{P}$ are strong: it must be a protocol that sends exactly 1 bit over every channel in every time step. Unfortunately, in a complete network, where the average number of messages sent per round in $\mathcal{P}$ is only $\tilde{O}(n)$, the multiplicative blowup can become $\tilde{\Omega}(n)$. This is much too high for practitioners who need noise-tolerant distributed algorithms.

This work is supported by the National Science Foundation grants CCF 1613772 and TWC 1318880.

In this paper, we take a first step toward designing an algorithm with costs that scale well for any network topology. In particular, if an adversary flips T bits, then our algorithm sends $\tilde{O}(L+T)$ bits, where L is the number of bits sent in $\mathcal{P}$. Importantly, our algorithm requires no *a priori* knowledge of T or L.

To obtain this result, we make two key simplifying assumptions over [37], [1], [12], [28]. First, we assume all channels are private and that the adversary does not know the private random bits of any player; this assumption is also in [15], [16]. Second, we assume that $\mathcal{P}$ is asynchronous: it runs correctly even with arbitrary message delays.

There exist many examples of asynchronous algorithms known for common problems that need to be solved in noisy networks including: leader election [17], minimum spanning tree [4], shortest paths [3], maximal independent set [2], stochastic gradient descent [43], and dominating set [44]. Our result allows using any asynchronous algorithm to design a new synchronous algorithm that is robust to channel noise. This new algorithm sends significantly fewer bits compared to prior work, which sends $\Omega(nL)$ bits [37], [1], [12], [28]. Additionally, when $T = \tilde{O}(L)$, the new algorithm sends only $\tilde{O}(L)$ bits.

A. Model Details

We now describe some details of our model. We refer the reader to [31], [42] for more details on synchrony and asynchrony in distributed computing.

Asynchrony of $\mathcal{P}$. Different from past work, we require that $\mathcal{P}$ runs correctly in an asynchronous network. Recall that an asynchronous network allows for adversarial delay of messages.

Synchrony of $\mathcal{P}'$. Following past work, we require that $\mathcal{P}'$ is run in a synchronous network. A synchronous network assumes that the amount of time needed to deliver a message over any channel is fixed and known. We define a *time step* as the amount of time that it takes to send one bit over a channel.

Binary Channels. In each time step, a bit value is set for every channel. This is the value received by any player listening on the channel. In any contiguous sequence of time steps where no player sends a bit, the adversary sets the bits, and pays a cost of 1 for each time it changes the value. For example, if there are 3 such contiguous time steps, the adversary can set the bit values to “000" or “111" at cost 0, or to “010" at cost 2. This cost model follows that in [16].

Additional Assumptions. As in prior work [37], [1], we assume that all players know the value n. However, we do not assume players know the values L, T and α. We assume private channels, but otherwise make no cryptographic assumptions. Finally, in the case where we want an error probability that is smaller than a polynomial in n, we require that all players know a desired error probability δ.

B. Our Result

Our main result is summarized in the following theorem. The *latency* of protocol $\mathcal{P}'$ is the total number of time steps that elapse until all processors in $\mathcal{P}'$ terminate. A *communication path* in an asynchronous simulation of $\mathcal{P}$ is a directed path over the players such that (1) any edge (u, v) in the path represents a message sent from player u to player v; and (2) for any two successive edges (u, v) and (v, w) in the path, the message associated with edge (u, v) was received by v prior to v sending the message associated with edge (v, w).

Theorem 1. *Let $\mathcal{P}$ be an asynchronous protocol for $n \geq 2$ players that sends a total of L bits. Let α be the average message length in $\mathcal{P}$. Let $\delta > 0$. Then we can create a synchronous protocol $\mathcal{P}'$ (via Algorithm 3 in Section III) with the following properties.*

1) *For any number T of adversarial bit flips, , $\mathcal{P}'$ succeeds with probability at least $1-\delta$. That is, each player stops after a finite number of time steps, and outputs a valid transcript of its run of $\mathcal{P}$.*
2) *The number of bits sent is $O\left(L\left(1+\frac{1}{\alpha}\log\left(\frac{nL}{\delta}\right)\right)+T\right)$.*
3) *The total latency is*

$$O\left(\max_p\left\{\Lambda_p\left(1+\frac{1}{\alpha}\log\left(\frac{n(L+T)}{\delta}\right)\right)+T_p\right\}\right)$$

where p ranges over all communication paths in the asynchronous simulation of $\mathcal{P}$, Λ_p is the latency of p, and T_p is the total number of bits flipped by the adversary on channels in p.

We make several observations about this result. First, $\mathcal{P}'$ always sends $\tilde{O}(L+T)$ bits, where the $\tilde{O}$ notation hides a logarithmic term in n, L and δ; if the desired δ is polynomially (or even sub-polynomially) small in n, then the $\tilde{O}$ notation hides a logarithmic (polylogarithmic) term just in n and L. Second, if α is $\Omega\left(\log(nL/\delta)\right)$, $\mathcal{P}'$ sends only $O(L+T)$ bits, and is thus within a constant factor of optimal. Finally, $\mathcal{P}'$ requires no *a priori* knowledge of L, T, α, or the network topology.[1] However, it does require all nodes know n, and that they agree on some desired δ.

C. High-level Overview

At a high level, our approach is as follows. For every channel that connects two players, each of these players runs two protocols for that channel, one for sending messages (Algorithm 1) and one for receiving messages (Algorithm 2).[2] Additionally, each player simulates $\mathcal{P}$ by running Algorithm 3. Algorithm 3 coordinates the actions of the player based on messages received through all instances that player is running of Algorithm 2. These actions may include sending messages via instances of Algorithm 1.

A naive idea is to run the 2-player algorithm of Dani et al. [16] over each communication channel. Unfortunately, this fails. In particular, the protocol of [16] assumes that a player sends a bit on the channel in each time step. However, in the multiparty problem, there may be many time steps where, for many channels, neither player on the channel sends a bit. This can happen because delay due to noise on one channel is holding up the protocol on all other channels, or simply because both players connected by a channel are waiting for messages from other players before using that channel.

We note that we do make use of two key ideas from [16]. First, we use *Algebraic Manipulation Detection (AMD) Codes* [14]. These codes enable detection of any bit flips in a code word with probability of error that is exponentially small in the number of bits added to the message word. Second, similarly to [16], we use error-correcting encoding of AMD code words to ensure that when we have to resend a message, our costs are linear in the number of bits flipped by the adversary (see Section III-A for details).

D. Technical Challenges

Below are the key techniques used in our algorithm.

Accepting Messages. Notifying others of errors when there are $n > 2$ players can be costly since errors can propagate through the network quickly. Thus, in contrast to [16], upon accepting a message, our protocol commits irrevocably to it.

This gives rise to two problems. First, we need to keep the total error probability small even when we do not know in advance the number of bits that will be sent in $\mathcal{P}$. To solve this, we carefully decrease error probabilities for each message sent over time, by slowly increasing codeword lengths. We do this in such a way that our total error probability is bounded by a sufficiently small geometric sum (see Section A-A).

Second, suppose a player receives the message m twice in a row from the same sender. We now must distinguish between two cases: (1) m is sent twice in a row in $\mathcal{P}$; and (2) the acknowledgement sent by the receiver of m was corrupted, and so the sender sent m again. We address this problem via a parity bit, similar to [19], which we use in our message send and receive protocols. See Section III-B for details.

Cost-Efficient Key Exchange. For $n > 2$ players, for any channel, there can be many time steps where neither player is sending on that channel. The adversary can try to forge messages during these time steps. To prevent this, we use a key-exchange protocol.

[1]We note that all prior interactive communication results assumes a priori knowledge of L.

[2]These protocols are run “in parallel" via multiplexing over the binary symmetric channel connecting the two players.

Naive key exchanges for our problem fail to be cost-efficient. For example, if only the sender sends a key, then during time steps when the sender is not sending, the adversary can forge messages from the sender. Thus, both the sender and the receiver must exchange keys.

Additionally, if the sender does not first send a message to initiate the key exchange, then the receiver does not know when to start receiving a message in $\mathcal{P}$. Then the receiver may unnecessarily send key-exchange messages on the channel even though the sender is not using the channel. Thus, our key exchange protocol has keys for sender and receiver, and also includes a key request phase.

Finally, some care must be taken to design the protocol so that (1) it works correctly when the sender is sending and the receiver is listening; and (2) when only the receiver is active, it learns this quickly, so that the number of bits sent by the players grows slowly with the number of bits flipped by the adversary. See Section III and Figure 1 for details.

Adjusting for Average Message Lengths. Finally, as an additional result, we show how to reduce resource costs when the average message length is high, even if it is not known in advance. We do this by dividing long messages into smaller chunks, which are sent in order. However, since we increase the codeword lengths over time, this gives rise to the following problem. If a chunk x of message m needs resending due to adversarial bit flips, then the resend in the following round will not only include the bits in x, but also some extra bits from m to make up for the increased codeword length. This introduces two challenges: (1) the sender of the message must decide how the message should be divided into chunks prior to sending it; and (2) the receiver of the message must eventually piece the chunks back together into the correct message.

On the sender side, we handle this by extracting the next chunk only when the current chunk has been correctly received, and no further resends are attempted. On the receiver side, we replace the local copy of the chunk received in round r with its resend received in round $r+1$ to account for the extra bits that the latter will contain. Additionally, the receiver must disambiguate a chunk resend from a repeated chunk using a parity bit. See Section C for details.

E. Paper Organization

The rest of this paper is organized as follows. In section II, we discuss some prior work that is related to our problem statement and results. In section III, we describe our main algorithm which handles the worst case where all messages in $\mathcal{P}$ are exactly 1 bit in length. We prove this algorithm is correct and analyze resource costs in Section A. In Section B, we describe our algorithm for the case where messages in $\mathcal{P}$ are of an arbitrary average size. We analyze this algorithm for resource costs and correctness in Section C. To better compare with past work, we discuss the notion of coding rate, and calculate it for our algorithm in Section IV. We conclude and give directions for future work in Section V.

II. Related Work

In this section, we survey related work on Interactive Communication and Rateless Codes. There has been significant interest in Interactive communication over the last several years, so for conciseness, we discuss only the most relevant work, with a focus on algorithms that are robust to adversarial noise. For a detailed survey of the field, we refer the reader to the excellent survey by Ran Gelles [21].

A. Interactive Communication ($n = 2$)

The problem of interactive communication was first posed by Schulman [39], [41], who describes a deterministic method for simulating interactive protocols on noisy channels with only a constant-factor increase in the total communication complexity. This initial work spurred vigorous interest in the area (see [7], [21] for some excellent surveys).

Schulman's scheme tolerates an adversarial noise rate of $1/240$, even when channels are public and the adversary knows all information. It critically depends on the notion of a *tree code* for which an exponential-time construction was originally provided. This exponential construction time motivated work on more efficient constructions [8], [36], [33]. There were also efforts to create alternative codes [22], [34]. Recently, elegant computationally-efficient schemes that tolerate a constant adversarial noise rate have been demonstrated [5], [24]. Additionally, a large number of results have improved the tolerable adversarial noise rate [6], [11], [25], [20], [9], as well as tuning the communication costs to a known, but not necessarily constant, adversarial noise rate [26].

B. Interactive Communication ($n > 2$)

Arbitrary Topologies. As noted previously, Rajagopalan and Schulman [37] have shown how to obtain a $O(\log n)$ blowup in the number of bits sent when $\mathcal{P}$ is an algorithm that sends exactly 1 bit over every channel in every time step. Their result holds when the fraction of bits corrupted by an adversary on any channel is constant, and even when all channels are public. Recently, Braverman et al [10] show this blowup is essentially tight in this model, by giving a lower bound for any interactive communication protocol running on a star network. In particular, they show that a blowup of $\Omega\left(\frac{\log n}{\log\log n}\right)$ is necessary when the fraction of corrupted bits is constant, even when the noise is stochastic.

Recent work by Censor-Hillel, Gelles and Haeupler [12] shows how to make any asynchronous distributed protocol robust against adversarial noise. The multiplicative bandwidth blowup for their protocols is $O(n\log^2 n)$. Critically, their robust protocol is asynchronous in contrast to our robust protocol which is synchronous. We can use the techniques in [12] to make our robust protocol asynchronous at the cost of a multiplicative bandwidth blowup of $O(n\log^2 n)$.

We note that all of the above protocols tolerate up to a $\Theta(1/n)$ fraction of message corruptions. This is optimal since, with public channels, if more than a $1/n$ fraction of messages are corrupted, the adversary can simply cut off all communication to a particular player. We discuss how we can tolerate a higher amount of noise with private channels in Section II-C.

Special Topologies. Alon et al. [1] present an algorithm that achieves a constant bit blowup for the special case of a complete network, or any network with constant mixing time. Hoza and Schulman [28] describe an algorithm that has constant bandwidth blowup for networks with $O(n)$ edges. However, their algorithm can have multiplicative bandwidth blowup of $\theta(n \log n)$ when the number of edges in the network is $\theta(n^2)$. We note that the algorithm of Jain, Kalai and Lewko [29] consider networks where one node is connected to every other node. They describe an algorithm with constant multiplicative bandwidth blowup for these networks. Finally, Gelles and Kalai [23] show that in a ring topology, logarithmic bandwidth blowup is necessary and sufficient (for asynchronous protocols).

We note that these previous results achieve an error probability that is exponentially small in the latency of $\mathcal{P}$.

C. How is our Model Different?

Most prior results for interactive communication give the *coding rate*, which is the blowup in the number of bits sent, as a function of the *error-rate*, which is the fraction of bits flipped. To better compare our result with past work, we make a similar calculation in Section IV.

Our work is not directly comparable to most past results because we do not quantify our results in terms of the fraction of messages that are corrupted, and more importantly, we assume that communication takes place over private channels.

The stronger assumption of private channels means that up to a $1/\log(nL)$ fraction of message bits can be corrupted, and our algorithm is still likely to succeed with a cost overhead that is only $O(\log(nL))$.

In our setting, the adversarial strategy of trying to cut off all communications to and from a single player requires corruption of much more than a $1/n$ fraction of bits. This is true since our protocol can detect the noise, and then increase the fraction of the total communication involving the beleaguered player.

Finally, to the best of our knowledge, all prior work on Interactive Communications assumes players know L in advance. We are able to remove this assumption of a priori knowledge of L.

D. Interactive Communication with Private Channels

Our paper builds on a result for 2-player interactive communication over a private channel by Dani et al [16]. Like our paper, their work tolerates an unknown by finite number of adversarial bit flips, T. They show that private channels are necessary in order to tolerate unknown T. Their algorithm assumes that both players know L. Their algorithm sends $L + O\left(\sqrt{L(T+1)\log L} + T\right)$ bits in expectation. We note that for $n = 2$ players, our protocol sends $O\left(L \log(L/\delta) + T\right)$ bits, which is larger for any choice of δ.

E. Rateless Codes

Rateless error correcting codes enable generation of potentially an infinite number of encoding symbols from a given set of source symbols, with the property that given any subset of a sufficient number of encoding symbols, the original source symbols can be recovered. These codes can tolerate an unknown amount of channel noise.

Fountain codes [32] and LT codes [35], [30], [27] are two classic examples of rateless codes. They employ feedback for stopping transmission [35], [30] and for error detection [27] at the receiver. Critically, the feedback channel is typically assumed to be noise free. We differ from this model in that we allow the adversary to flip bits on the feedback channel. Additionally, we tolerate bit flips, while most rateless codes tolerate only bit erasures. Not surprisingly, these codes incur a smaller constant blowup in the number of bits sent when compared to our results.

III. Our Algorithm

We discuss the overall idea and technical aspects of our main algorithm in this section. The detailed steps are presented in Algorithm 3 in Section III-D, which uses Algorithms 1 and 2 as subroutines. We also provide a flowchart to illustrate the different steps for communication over a single channel in Figure 1.

For now, we assume that $\mathcal{P}$ consists of one-bit messages. We remove this assumption to handle arbitrary message lengths in Section B.

A. Notation and Definitions

Some helper functions and notation used in our algorithm as follows. For a string s, we use the notation $s[i]$ to denote the i^{th} bit of s and $s[i,j]$ to denote the substring $(s[i], s[i+1], \ldots, s[j-1])$. We let $|s|$ denote the length of string s, and use the conventions that $s = (s[0], s[1], \ldots, s[|s|-1])$, and for $j > |s|$, $s[i,j] = s[i,|s|]$. We use $\oplus$ to denote the XOR operation on bit strings. We assume that all logarithms used in this paper are of base two.

Algebraic Manipulation Detection Codes. Our algorithm makes use of Algebraic Manipulation Detection (AMD) codes from [14]. For a given $\eta > 0$, these codes provide three functions: `amdEnc`, `amdDec` and `IsCodeword`. The function `amdEnc` (m, η) creates an AMD encoding of a message m. The function `IsCodeword` (m, η) takes a message m and returns true if and only if there exists some message m' such that `amdEnc` $(m', \eta) = m$. The function `amdDec` (m, η) takes a message m, checks if `IsCodeword` (m, η) is true and if so, returns a message m' such that `amdEnc` $(m', \eta) = m$. Else, it returns a dummy message, denoted $\perp$, indicating failure. AMD codes do not make any cryptographic assumptions and only require that the communication happens on private channels.

We summarize the results from [14] to highlight the important properties of these functions in the following lemma.

Theorem 2. *There exist functions `amdEnc`, `amdDec` and `IsCodeword`, such that for any $\eta \in \left(0, \frac{1}{2}\right]$ and any bit string m of length x:*

1) $\texttt{amdEnc}(m,\eta)$ *is a string of length* $x + 2\log\left(\frac{1}{\eta}\right)$.
2) $\texttt{IsCodeword}(\texttt{amdEnc}(m,\eta),\eta) = $ *true and*

$$\texttt{amdDec}(\texttt{amdEnc}(m,\eta),\eta) = m.$$

3) *For any bit string* $s \neq 0$ *of length* x*, we have*

$$\Pr(\texttt{IsCodeword}(\texttt{amdEnc}(m,\eta) \oplus s,\eta)) \leq \eta.$$

With respect to these properties, we will refer to η as the *strength* or *security* of the encoding or *AMD failure probability*, depending on the context.

$\frac{1}{3}$-Error-correcting Codes. These codes enable us to encode a message so that it can be completely recovered if the adversary corrupts up to a third of the total number of bits in the message. We will denote the encoding and decoding functions by $\texttt{ecEnc}$ and $\texttt{ecDec}$, respectively. The following theorem, established by the results in [38], gives the properties of these functions.

Theorem 3. *There exists a constant* $C_e > 0$ *such that for any message* m*, we have* $|\texttt{ecEnc}(m)| = C_e|m|$*. Moreover, if* m' *differs from* $\texttt{ecEnc}(m)$ *in at most one third of its bits, then* $\texttt{ecDec}(m') = m$*.*

We observe that the linearity of $\texttt{ecEnc}$ and $\texttt{ecDec}$ ensures that when the error correction is composed with the AMD code, the resulting code has the following properties:

1) If at most a third of the bits of the message are flipped, then the original message can be uniquely reconstructed by rounding to the nearest codeword in the range of $\texttt{ecEnc}$.
2) Even if an arbitrary set of bits is flipped, the probability of the change not being recognized is at most η, i.e. the same guarantee as for the plain AMD codes.

This is because the error-correcting code is linear, so that when noise η is XOR'ed by the adversary to the codeword x, effectively what happens is that the decoding function rounds the noise to the nearest codeword. Thus, $\texttt{ecDec}(x \oplus \eta) = \texttt{ecDec}(x) \oplus \texttt{ecDec}(\eta) = m \oplus \texttt{ecDec}(\eta)$, where m is the AMD-encoded message. But now $\texttt{ecDec}(\eta)$ is an obliviously selected string added to the AMD-encoded codeword, and hence the result is very unlikely to be a valid message unless $\texttt{ecDec}(\eta) = 0$.

Silence. We define the function $\texttt{IsSilence}(s)$ to return true iff the string s has fewer than $|s|/3$ bit alternations. We also define $\mathscr{S}_\ell = \{s \in \{0,1\}^\ell \mid \texttt{IsSilence}(s)\}$. We drop the subscript when ℓ is clear from the context.

B. Algorithm Overview

Our algorithm proceeds in rounds, each of which consists of the following steps. For convenience, our presentation assumes two bidirectional channels between each pair of (neighboring) players, one for each player to initiate a round with the other. We can simulate these two channels by multiplexing over a single channel, at the cost of a factor 2 increase in the number of time steps.

1) If u has a message for v, it initiates a message exchange by asking v for a key. We use the keyword KEY? to denote this request for the key.
2) Upon receipt of this key, u sends the message along with the key.
3) v terminates the message exchange upon successful authentication and retrieval of the message.
4) u terminates the message exchange upon hearing silence from v.

This goes on until all the messages in $\mathcal{P}$ have been communicated to the intended recipients.

Rounds. For each message m in $\mathcal{P}$ that needs to be sent from some player u to his neighbor v, we communicate m through a sequence of exchanges between players u, referred to as *Alice*, and v, referred to as *Bob*, in $\mathcal{P}'$ using Algorithms 1 and 2, respectively. As mentioned above, the sequence of time steps corresponding to steps 1-4 of the algorithm overview constitute what we call a *round*. Thus, each round of $\mathcal{P}$ consists of exactly four *words*, one for each of the steps 1-4. The length of each word in round r is denoted w_r. This depends on the round number r and is therefore a function of the current time step, which can be computed independently by each player using the global clock. We make w_r gradually increase with r (the exact dependence will be provided shortly).

Since each message in $\mathcal{P}$ is just a single bit, it takes exactly one round to be communicated in $\mathcal{P}'$, if no successful corruption happens, and more otherwise. If a round for some message m is corrupted, we attempt resending m in the subsequent round, which provides higher security due to a larger word length.

At the beginning of each round, Bob must listen for a key request during the first w_r time steps. If no valid request is received, he idles until the start of the next round. Thus Bob is active in every round. For her part, Alice only participates in a round if, in $\mathcal{P}$, she has a message to send to Bob.

Parity Bit. Observe that it is possible that Alice and Bob have different views on whether a particular round was successful. More concretely, if Bob encounters two progressive rounds that contain the same message, with no silent round in between, he needs to distinguish between whether Alice is resending the message or whether Alice's next message happens to be the same as the previous one. To disambiguate these cases, Alice appends a bit b to each message m that she sends to Bob, where b is the parity of the index of m. We will refer to b as the *parity bit* for m and use the notation (m,b) to denote this augmented message.

Word Generation. Both Alice and Bob generate their words for round r using a function $\mathscr{E}_r(x,k)$, described below, which returns the encoding of the word's content x using the key k based on the security settings for round r. Here x may be the message m from $\mathcal{P}$, a special keyword KEY? used by Alice to request Bob's key, or Bob's key for the round. The key k, for round r, is a string of length $2\left\lceil \log\frac{4n\pi r}{\sqrt{\delta}}\right\rceil$ bits. This key length, and other corresponding encoding parameters, are chosen to

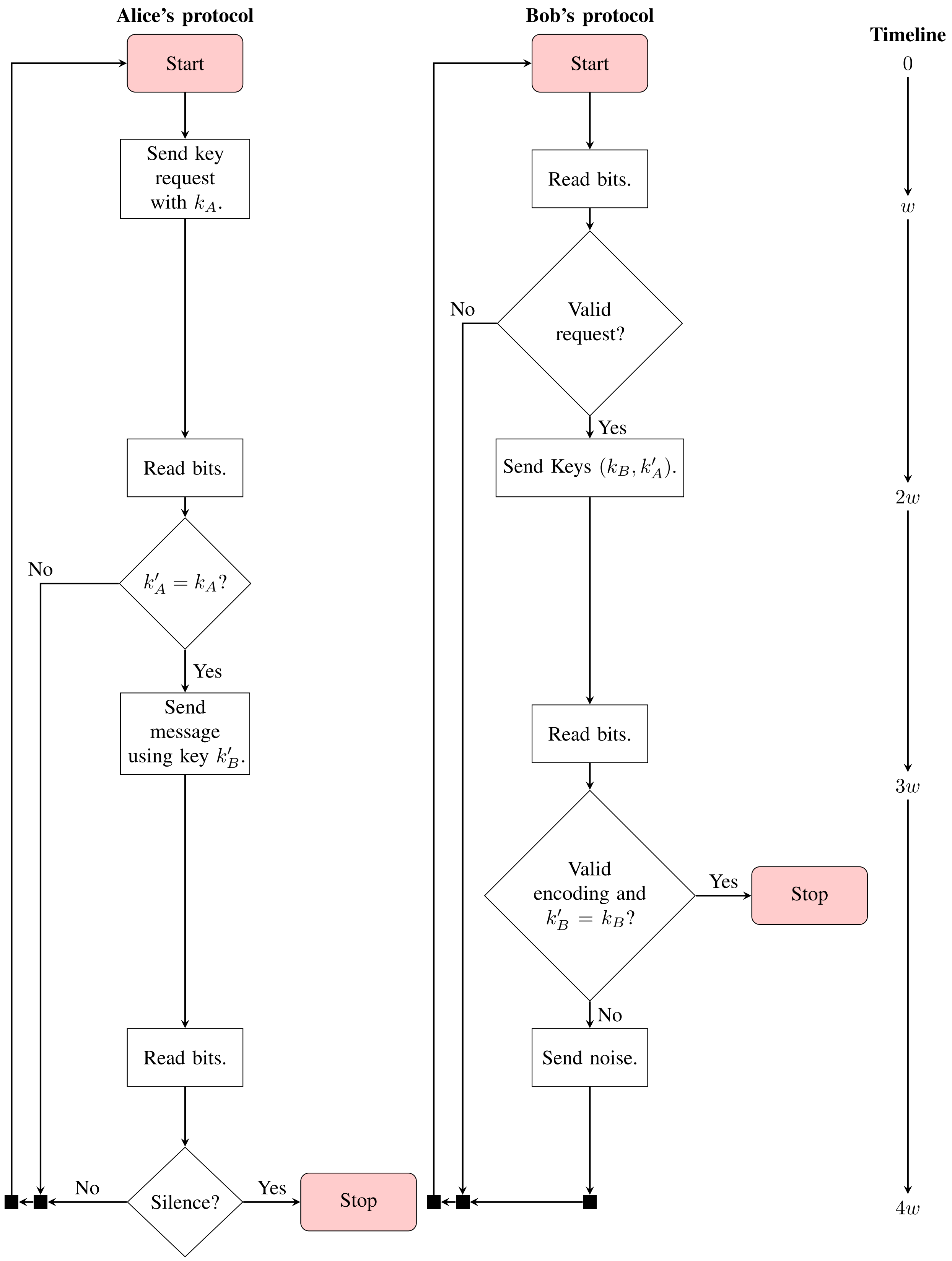

Fig. 1: Flowchart for Alice and Bob during each round. Here w denotes the word length in the current round.

ensure the failure events for our algorithm occur within the given error tolerance, δ (See Section A-A for details).

In Alice's first call to $\mathscr{E}$, k is a random string. In Bob's call, k is the key he received from Alice in the previous word, but x is a random string of the appropriate length. In Alice's second call, k is the key she received from Bob in the previous word. Alice and Bob generate fresh random keys for each round in which they desire to send a message.

The function $\mathscr{E}_r(x, k)$ is formally defined as follows. The function first appends the content x with a key k of length $\kappa_r = 2\left\lceil \log \frac{4n\pi r}{\sqrt{\delta}} \right\rceil$ bits. Then, it encodes this concatenated string of bits with an AMD code using $\eta_r = \frac{\delta}{2n^2\pi^2 r^2}$. Then, it encodes this AMD codeword with a (1/3)-error-correcting code. Finally, it pads $38\left\lceil \log \frac{2n\pi r}{\sqrt{\delta}} \right\rceil$ random bits to the ECC codeword. These random bits are added to ensure that players are unlikely to confuse words with silence and that the adversary is unlikely to flip bits of a word to forge silence. As a result, the word length for round r is given as $w_r = O\left(\log \frac{nr}{\delta}\right)$.

Similar to $\mathscr{E}_r(x, k)$, we define a function $\mathscr{D}_r(m)$ which returns either a pair (x', k') such that $\mathscr{E}_r(x', k') = m$ or returns $(\perp, \perp)$, if no such (x', k') exists. It begins by stripping off the padding at the end of m to obtain a shorter string m'. Then it decodes the error correction, computing $m'' = \texttt{ecDec}(m')$. If $\texttt{IsCodeword}(m'', \eta_r)$, then $\mathscr{D}$ outputs $\texttt{amdDec}(m'', \eta)$, otherwise it returns $(\perp, \perp)$.

Failure Events: There are certain events which may cause our algorithm to fail. More specifically, it is possible that the adversary (1) converts one AMD codeword to another, so that the decoded content is different from the content intended; (2) converts a non-silence word into silence; and (3) correctly guesses some player's key and uses it to communicate bits that are not in $\mathcal{P}$. We will discuss these failure events in detail in Section III-E and analyze their probabilities of occurrence in Section A-A.

C. Round Details

We will now describe various scenarios for what may happen in a round during the execution of the algorithm, for a particular bidirectional channel. There are $O(n^2)$ such channels, and the same round may have different scenarios enacted on it on different channels. Moreover, the views of Alice and Bob may differ on which scenario was enacted. For the sake of simplicity, we assume that none of the failure events as described above happen for the remainder of this section.

Silent Rounds: These are rounds in which Bob hears silence when he listens to the first word of the round on the channel. Since the adversary cannot tamper with messages from Alice to look like silence, such a round can happen only when Alice had no message for Bob. Thus, nothing further happens in the round and both Alice and Bob stay silent. This way, both views agree that the round was silent.

Progressive Rounds: These are rounds in which the number of bits flipped by the adversary is small enough that it is handled by the error-correction codes, so that encoded messages

Algorithm 1 Message exchange algorithm for the sender. `Send-Message` is only called at the beginning of a round.

```
1:  procedure SEND-MSG($m$)
2:      ▷ $b$ is a persistent variable for the parity bit. On the first call to SEND-MSG, $b$ is set to 0. On subsequent calls it is whatever it was set to on the previous call.
3:      while true do
4:          Generate random key $k_A$ of length $\kappa_r$   ▷ $r$ is the current round number..
5:          Send $\mathscr{E}_r(\texttt{KEY?}, k_A)$.
6:          $M_1 \leftarrow w_r$ bits from the receiver.
7:          if $\texttt{IsSilence}(M_1)$ then   ▷ assume the receiver has already terminated.
8:              Stay silent for $2w_r$ time steps and return
9:          else
10:             $(x, k) \leftarrow \mathscr{D}_r(M_1)$
11:             if $k \neq k_A$ then
12:                 Stay silent for $2w_r$ time steps.
13:             else
14:                 $k_B \leftarrow x$
15:                 Send $\mathscr{E}_r((m, b), k_B)$.
16:                 $M_2 \leftarrow w_r$ bits from the receiver.
17:                 if $\texttt{IsSilence}(M_2)$ then
18:                     $b \leftarrow \neg b$ and return
```

Algorithm 2 Message exchange algorithm for the receiver. Receive-Message is only called at the beginning of a round.

```
1:  procedure RECEIVE-MSG( )
2:      ▷ $\hat{b}$ is a persistent variable for the parity bit. On the first call to RECEIVE-MSG, $\hat{b}$ is set to 0. On subsequent calls it is whatever it was set to on the previous call.
3:      $M_1' \leftarrow w_r$ bits from the sender.
4:      if $\texttt{IsSilence}(M_1')$ then
5:          Stay silent for $3w_r$ time steps.   ▷ $r$ is the current round number.
6:      else
7:          $(x', k') \leftarrow \mathscr{D}_r(M_1')$
8:          if $x' \neq \texttt{KEY?}$ then
9:              Send noise for $w_r$ time steps.
10:             Stay silent for $2w_r$ time steps.
11:         else
12:             $k_A \leftarrow k'$.
13:             Generate random key $k_B$ of length $\kappa_t$.
14:             Send $\mathscr{E}_r(k_B, k_A)$.
15:             $M_2' \leftarrow w_r$ bits from the sender.
16:             $(x'', k'') \leftarrow \mathscr{D}_r(M_2')$
17:             if $k'' \neq k_B$ then
18:                 Send noise for $w_r$ time steps.
19:             else
20:                 $(m', b') \leftarrow x''$
21:                 if $b' \neq \hat{b}$ different from last message then
22:                     Set $\hat{b} \leftarrow b'$ and record the message $m'$ from the sender.
23:                 Stay silent for $w_r$ time steps.
```

are successfully recovered upon receipt. Furthermore, when Bob is silent (after successful authentication of the message from Alice), the adversary is unable to set sufficiently many bits on the channel from Bob to make Alice believe otherwise. This allows Alice to decide that Bob has successfully received her message. Hence, when the round ends, both Alice and Bob agree that the round was successful.

Corrupted Rounds: These are rounds in which the adversary is able to successfully corrupt one or more of the words in the round. The following cases are possible.

1) Alice is silent, but the adversary sends Bob a key request. Following this, Bob samples a key and sends it to Alice. But she is not expecting a message from Bob and hence, remains silent. At this point the adversary can say whatever he wants on the channel. However, since he does not know Bob's key[3], he cannot authenticate his message. Thus, Bob receives an invalid communication and responds with noise. Again, since Alice is not expecting any message from Bob at this point, she remains silent and does nothing. Thus, *such a round is corrupted in Bob's view, but silent in Alice's*.[4]
2) The adversary corrupts Alice's key request. Bob, upon receipt of this invalid key request, remains silent in the rest of the round. The adversary may now say whatever he wants on the channel from Bob. In particular, he may try to impersonate Bob and send Alice a key. However, since he cannot reproduce Alice's key from the request that she originally sent to Bob, the adversary is unable to authenticate himself as Bob. Thus, Alice receives an invalid communication, and stays silent until the end of the round.
3) Bob receives Alice's key request, but the adversary corrupts Bob's reply, which contains his key. However, since the adversary cannot corrupt one AMD codeword into another, Alice is unable to decode the message into a key for Bob, causing her to remain silent for the remainder of the round. Again, the adversary cannot forge a message from Alice since he cannot reproduce Bob's key.
4) The adversary is inactive for the first half of the round until Alice receives Bob's key. After that, the adversary corrupts Alice' communication of her message to Bob. This will disable Bob to decode the message at his end and consequently, he injects noise into the channel. Since the adversary cannot convert this noise into silence, Alice knows that the round has failed, and of course, so does Bob.
5) The round succeeds all the way to the point where Bob decides that the round is successful and remains silent. The adversary then injects noise onto the channel causing Alice to think the round has failed. Thus, *such a round is corrupted in Alice's view but progressive in Bob's*. However, this is not a problem because when Alice resends the message in the following round, she keeps the parity bit unchanged to indicate to Bob that this is a resend (and not a fresh message). Hence, Bob will receive the message again in this round. At this stage, of course, Bob realizes that the previous round was in fact, corrupted, and disregards the message he believed was successfully received in the previous round. There is no other difference in Bob's future actions.

D. The protocol $\mathcal{P}'$

We now present our main protocol $\mathcal{P}'$, described by Algorithm 3.

As shown in [31] Chapter 14.1.1, any asynchronous distributed algorithm can be represented as an I/O automaton with certain properties. We make use of this definition to represent $\mathcal{P}$. This formulation will be helpful in proving the required security properties of our protocol in Section A-B.

We assume that for each player u in the network, $\mathcal{P}$ provides an I/O automaton $\mathcal{P}_u$ with the following properties. [5]

- $\mathcal{P}_u$ has a single *initial* state.
- $\mathcal{P}_u$ has some subset of states that are *termination* states. Each termination state may have a value for u to *output*.
- There is a set of *transition relations*, each from one state to another state, each labeled with an *action*, where this action may be either an *input* action (e.g. receiving a message) or an *output* action (e.g. sending a message). These transitions satisfy the property for every state s, for every possible input action, a, there is a transition from s to some other state that is labelled with a.

E. Failure Events

As mentioned before, there are certain events that can cause catastrophic failure from which $\mathcal{P}'$ cannot recover. These events are described as follows.

1) The adversary's bit flips happen to convert an AMD codeword into another valid AMD codeword. In this case, the decoded content differs from the intended content, resulting in either authentication failure (when the content was Bob's key) or incorrect simulation of $\mathcal{P}$ (when the content was Alice's message).
2) The adversary's bit flips on some (non-silent) word are such that the resulting word looks like silence to its recipient. If the noise sent by Bob in line 18 of Algorithm 2 to request a resend is converted to silence, then Alice incorrectly assumes that Bob has received her message and stops transmitting it. This results in an incorrect transcript of $\mathcal{P}$. Other words being converted to silence could result in a player being silent on the following word, which in conjunction with guessing the

[3]Recall our assumption about private channels and the fact that the adversary is oblivious to the private random bits of the players.

[4]Note that Bob may realize that the round was silent in Alice's view at a later stage, but we still account for this as a corrupted round, since Bob has already incurred a cost for the corruption.

[5]We use an I/O automaton to capture subtleties of simulating an asynchronous protocol. For example, when player x sends a message to player y, we must show that either player y eventually receives this message, or that player y terminates in $\mathcal{P}$ before receipt of the message.

Algorithm 3 Protocol $\mathcal{P}'$, run at each node u

```
1: $s \leftarrow$ initial state of $\mathcal{P}_u$
2: while $s$ is not a termination state do
3:     if there is a transition relation from state $s$ to state $s'$ in $\mathcal{P}_u$ labeled with an output action to send message $m$ to neighbor $v$ then
4:         Schedule SEND-MSG($m$) to run on the channel to $v$.
           If no call to SEND-MSG is currently running on the channel, begin immediately.
           Otherwise, set up the call to begin as soon as all currently scheduled SEND-MSG
           calls on this channel have finished running.
5:         Transition to state $s'$ in $\mathcal{P}_u$; $s \leftarrow s'$.
6:     else
7:         repeat
8:             for each neighbor $v$, in parallel run RECEIVE-MSG() on the channel from $v$.
9:         until the RECEIVE-MSG calls have recorded some non-empty set $S$ of messages
10:        for each message $m$ in $S$ do
11:            Let $s'$ be the target state on the transition relation from $s$ with input action $m$.
12:            Transition to state $s'$ in $\mathcal{P}_u$; $s \leftarrow s'$.
13: Set player $u$'s output based on the termination state.
14: Continue executing any remaining scheduled SEND-MSG calls until all have returned.
```

key (see below), could result in the adversary being able to say whatever he wants on the channel.

3) In a round when Alice is silent in $\mathcal{P}$, the adversary can forge a key request on her channel to Bob. Ordinarily, this is not a problem because when Bob responds with his key, the adversary cannot read it, and therefore cannot send Bob an authenticated message. However, if he happens to guess Bob's key, then he can send Bob a message purporting to be from Alice, resulting in an incorrect simulation of $\mathcal{P}$.

We will show in Section A-A that our choice of word lengths and encoding strengths over different rounds ensure that the probability of such a catastrophic failure, caused by either (or any combination of) of these failure events over the entire run of the algorithm is within the given error tolerance, δ.

IV. Coding Rate of our Algorithm

In most prior work it is assumed that the noise rate of the channel(s) is known and may be used as a parameter in the design of the algorithm. The parameter of interest then is the rate of the designed code, or *coding rate*. The coding rate of an interactive communication algorithm is the number of bits sent in $\mathcal{P}'$ divided by the number of bits sent in $\mathcal{P}$.

In our work we do not assume that the noise rate is known in advance, but only require that the adversary flip a finite number of bits. In order to compare our result with other work, we compute the coding rate of our algorithm as a function of the *a posteriori* noise rate. Such a comparison is only meaningful when the adversary's total budget is less than L, the length of $\mathcal{P}$, so for this section we will assume that $T < L$.

Let L' denote the length of $\mathcal{P}'$. Theorem 1 states that our algorithm achieves $L' \leq CL\left(1 + \frac{1}{\alpha}\log\left(\frac{nL}{\delta}\right)\right) + CT$ for some constant C, where δ is the permissible failure probability for the algorithm and α is the average message length in $\mathcal{P}$. Furthermore, since $T < L$ this translates to the absolute upper bound

$$L' \leq CL\left(2 + \frac{1}{\alpha}\log\left(\frac{nL}{\delta}\right)\right) \quad (1)$$
$$\leq CL\left(2 + \frac{1}{\alpha}\log\left(\frac{n(L+T)}{\delta}\right)\right) \quad (2)$$
$$\leq CL\left(2 + \frac{1}{\alpha}\log\left(\frac{2nL}{\delta}\right)\right) \quad (3)$$

Let $\varepsilon = T/L'$ be the *a posteriori* noise rate. Then $T = \varepsilon L'$. Making this substitution for T in (2), and using (3) to bound the L' inside the log, we have $L' \leq CL\left(2 + \frac{1}{\alpha}\log\left(\frac{n\left(L+\varepsilon CL\left(2+\frac{1}{\alpha}\log\left(\frac{2nL}{\delta}\right)\right)\right)}{\delta}\right)\right)$. Dividing by L, we get $\frac{L'}{L} \leq 2C + \frac{C}{\alpha}\log\left(\frac{n\left(L+\varepsilon CL\left(2+\frac{1}{\alpha}\log\left(\frac{2nL}{\delta}\right)\right)\right)}{\delta}\right)$ which the same as saying that

$$\frac{L'}{L} \leq 2C + \frac{C}{\alpha}\left[\log(nL/\delta) + \log\left(1 + \varepsilon\left(2C + \frac{C\log(2nL/\delta)}{\alpha}\right)\right)\right].$$

Finally, using $\log(1+x) \leq x$ on the second logarithm in this expression, we get $\frac{L'}{L} \leq \left(2C + \frac{C\log(2nL/\delta)}{\alpha}\right)\left(1 + \frac{\varepsilon C}{\alpha}\right)$.

To summarize, for the worst case where $\alpha = 1$, the above shows that we achieve a coding rate that increases linearly with the noise rate ε and with the logarithm of nL/δ. In particular, the coding rate is $O((1+\varepsilon)\log(nL/\delta))$. For arbitrary α, we achieve a coding rate of $O((1+\varepsilon/\alpha)\log(nL/\delta)/\alpha)$.

V. Conclusion and Future Work

We have described the first algorithm in interactive communication for n players that deals with the case of unknown number of bits sent by the protocol, while tolerating an unbounded but finite amount of noise. Against an adversary that flips T bits, given an $\delta \in (0,1)$, our algorithm compiles a noise free protocol $\mathcal{P}$ that sends L bits into a robust protocol $\mathcal{P}'$ that succeeds with probability $1-\delta$, and upon successful termination, sends $O\left(L\left(1 + \frac{1}{\alpha}\log\left(\frac{n(L+T)}{\delta}\right)\right) + T\right)$ bits, where α is the average message length in $\mathcal{P}$. The blowup in the number of bits is constant for long messages in $\mathcal{P}$ and within logarithmic factors of the optimal, otherwise.

Several open problems remain including the following. First, can we adapt our results to interactive communication where $\mathcal{P}$ is a synchronous protocol? Second, can we handle an unknown amount of stochastic noise more efficiently, while making no assumption on the value of L or T? Finally, for any algorithm, what is the minimum number of private random bits required to be hidden from the adversary to achieve robustness?

References

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

[2] Baruch Awerbuch, Lenore J Cowen, and Mark A Smith. Efficient asynchronous distributed symmetry breaking. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 214–223. ACM, 1994.

[3] Dimitri P Bertsekas and Robert G Gallager. Distributed asynchronous bellman-ford algorithm. *Data networks*, page 4, 1987.

[4] Lélia Blin, Fadwa Boubekeur, and Swan Dubois. A self-stabilizing memory efficient algorithm for the minimum diameter spanning tree under an omnipotent daemon. *Journal of Parallel and Distributed Computing*, 117:50–62, 2018.

[5] Zvika Brakerski and Yael Tauman Kalai. Efficient Interactive Coding against Adversarial Noise. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 160–166, 2012.

[6] Zvika Brakerski and Moni Naor. Fast Algorithms for Interactive Coding. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 443–456, 2013.

[7] Mark Braverman. Coding for Interactive Computation: Progress and Challenges. In *Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1914–1921, Oct 2012.

[8] Mark Braverman. Towards Deterministic Tree Code Constructions. In *Proceedings of Innovations in Theoretical Computer Science Conference (ITCS)*, pages 161–167, 2012.

[9] Mark Braverman and Klim Efremenko. List and Unique Coding for Interactive Communication in the Presence of Adversarial Noise. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 236–245, 2014.

[10] Mark Braverman, Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Constant-rate Coding for Multiparty Interactive Communication is Impossible. In *Electronic Colloquium on Computational Complexity (ECCC), TR15-197*, 2015.

[11] 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)*, pages 159–166, 2011.

[12] Keren Censor-Hillel, Ran Gelles, and Bernhard Haeupler. Making asynchronous distributed computations robust to channel noise. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 50:1–50:20, 2018.

[13] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[14] Ronald Cramer, Yevgeniy Dodis, Serge Fehr, Carles Padró, and Daniel Wichs. Detection of Algebraic Manipulation with Applications to Robust Secret Sharing and Fuzzy Extractors. In *Advances in Cryptology–EUROCRYPT 2008*, pages 471–488. Springer, 2008.

[15] Varsha Dani, Thomas P. Hayes, Mahnush Movahedi, Jared Saia, and Maxwell Young. Interactive Communication with Unknown Noise Rate. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, 2015.

[16] Varsha Dani, Thomas P. Hayes, Mahnush Movahedi, Jared Saia, and Maxwell Young. Interactive communication with unknown noise rate. *Information and Computation*, 2018.

[17] Dariusz Dereniowski and Andrzej Pelc. Leader election for anonymous asynchronous agents in arbitrary networks. *Distributed Computing*, 27(1):21–38, 2014.

[18] Devdatt P Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

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

[20] Matthew Franklin, Ran Gelles, Rafail Ostrovsky, and Leonard Schulman. Optimal Coding for Streaming Authentication and Interactive Communication. *IEEE Transactions on Information Theory*, 61(1):133–145, 2015.

[21] Ran Gelles et al. Coding for interactive communication: A survey. *Foundations and Trends® in Theoretical Computer Science*, 13(1–2):1–157, 2017.

[22] Ran Gelles, Ankur Moitra, and Amit Sahai. Efficient and Explicit Coding for Interactive Communication. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 768–777, Oct 2011.

[23] Ran Gelles and Yael T Kalai. Constant-rate interactive coding is impossible, even in constant-degree networks. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 67. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[24] Mohsen Ghaffari and Bernhard Haeupler. Optimal Error Rates for Interactive Coding II: Efficiency and List Decoding. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 394–403. IEEE, 2014.

[25] Mohsen Ghaffari, Bernhard Haeupler, and Madhu Sudan. Optimal Error Rates for Interactive Coding I: Adaptivity and Other Settings. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 794–803, 2014.

[26] Bernhard Haeupler. Interactive Channel Capacity Revisited. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 226–235. IEEE, 2014.

[27] Morteza Hashemi and Ari Trachtenberg. Near Real-time Rateless Coding with a Constrained Feedback Budget. In *Proceedings of the Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 529–536. IEEE, 2014.

[28] William M Hoza and Leonard J Schulman. The adversarial noise threshold for distributed protocols. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 240–258. SIAM, 2016.

[29] Abhishek Jain, Yael Tauman Kalai, and Allison Bishop Lewko. Interactive coding for multiparty protocols. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 1–10. ACM, 2015.

[30] Michael Luby. LT Codes. In *Proceedings of the IEEE Foundations of Computer Science (FOCS)*, 2002.

[31] Nancy A Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[32] David JC MacKay. Fountain Codes. In *IEE Proceedings - Communications*, volume 152, pages 1062–1068. IET, 2005.

[33] Cristopher Moore and Leonard J. Schulman. Tree Codes and a Conjecture on Exponential Sums. In *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 145–154, 2014.

[34] Rafail Ostrovsky, Yuval Rabani, and Leonard J. Schulman. Error-Correcting Codes for Automatic Control. *IEEE Transactions on Information Theory*, 55(7):2931–2941, 2009.

[35] Ravi Palanki and Jonathan S Yedidia. Rateless Codes on Noisy Channels. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 37–37. Citeseer, 2004.

[36] Marcin Peczarski. An Improvement of the Tree Code Construction. *Information Processing Letters*, 99(3):92–95, 2006.

[37] Sridhar Rajagopalan and Leonard Schulman. A coding theorem for distributed computation. In *Proceedings of ACM Symposium on Theory of computing (STOC)*, pages 790–799. ACM, 1994.

[38] Irving S Reed and Gustave Solomon. Polynomial Codes over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[39] Leonard J. Schulman. Deterministic Coding for Interactive Communication. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 747–756, 1993.

[40] Leonard J. Schulman. Coding for Interactive Communication. *IEEE Transactions on Information Theory*, 42(6):1745–1756, 1996.

[41] L.J. Schulman. Communication on Noisy Channels: A Coding Theorem for Computation. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 724–733, Oct 1992.

[42] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 2000.

[43] John Tsitsiklis, Dimitri Bertsekas, and Michael Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE transactions on automatic control*, 31(9):803–812, 1986.

[44] Peng-Jun Wan, Khaled M Alzoubi, and Ophir Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. In *INFOCOM 2002. Twenty-First annual joint conference of the IEEE computer and communications societies. Proceedings. IEEE*, volume 3, pages 1597–1604. IEEE, 2002.

Appendix A
Analysis of Algorithm 3

We now analyze the failure probability and total number of bits sent in $\mathcal{P}'$. We begin by computing the failure probability for the algorithm, by considering the three bad events as before, and then take a union bound over all the rounds. We will then prove that the algorithm is correct and terminates in finitely many time steps. Finally, we compute an upper bound on the expected number of bits the algorithm sends.

A. Probability of Failure

We define three bad events for a round.

1) AMD Failure : The adversary is able to flip the bits to convert the message into another valid word. In this case, the affected players end their rounds either with authentication failures or with knowledge of bits which are not in $\mathcal{P}$.
2) Conversion to Silence : The adversary is able to flip bits in such a way that some player's random bits look like silence to his neighbor, resulting in the latter ending his round without the knowledge of this failure.
3) Key Installation : The adversary installs a correct key from Bob when Alice is silent (to successfully simulate line 15 of Algorithm 1) or installs a correct key from Alice when Bob is silent (to successfully simulate line 14 of Algorithm 2).

We bound the probabilities of each of these failure events by $\delta/3$ in Lemmas 1, 3 and 4, respectively. Finally, we use a union bound in Lemma 5 to eventually prove that our algorithm succeeds with probability at least $1-\delta$.

Lemma 1. *In $\mathcal{P}'$, AMD Failure occurs with probability at most $\delta/3$.*

Proof. Recall that in any round, four words are exchanged between any pair of players, and that at most $\binom{n}{2} < n^2$ pairs of players may be exchanging words with each other. Furthermore, in a given round r, the AMD failure probability for a single word is set to be at most $\eta_r = (n\pi r)^{-2}\,\delta/2$, as discussed in Section III-B. Now, for $1 \le k \le 4, 1 \le i \ne j \le n$ and $r \ge 1$, define $\xi_{i,j,k,r}$ to be the event that in round r, AMD failure occurs in the k^{th} word exchanged between players i and j. Then, $\Pr(\xi_{i,j,k,r}) \le (n\pi r)^{-2}\,\delta/2$. Hence, by a union bound, we get $\Pr\left(\bigcup_{i,j,k,r} \xi_{i,j,k,r}\right) \le \sum_{k=1}^{4}\sum_{\substack{i,j=1\\ i\ne j}}^{n}\sum_{r\ge 1}\frac{\delta}{2(n\pi r)^2} \le \frac{2\delta}{\pi^2}\sum_{r\ge1}\frac{1}{r^2} = \frac{\delta}{3}$. □

Lemma 2. *For $b \ge 95$, the probability that a b-bit string sampled uniformly at random from $\{0,1\}^b$ has fewer than $b/3$ bit alternations is at most $e^{-b/19}$.*

Proof. Let s be a string sampled uniformly at random from $\{0,1\}^b$, where $b \ge 95$. Denote by $s[i]$ the i^{th} bit of s. Let X_i be the indicator random variable for the event that $s[i] \ne s[i+1]$, for $1 \le i < b$. Note that all X_i's are mutually independent. Let X be the number of bit alternations in s. Clearly, $X = \sum_{i=1}^{b-1} X_i$, which gives $\mathbb{E}(X) = \sum_{i=1}^{b-1}\mathbb{E}(X_i)$, using the linearity of expectation. Since $\mathbb{E}(X_i) = 1/2$ for all $1 \le i < b$, we get $\mathbb{E}(X) = (b-1)/2$. Using a multiplicative version of Chernoff's bound (see [18]), we have that for $0 \le t \le \sqrt{b-1}$,

$$\Pr\left(X < \frac{b-1}{2} - \frac{t\sqrt{b-1}}{2}\right) \le e^{-t^2/2}.$$

To obtain $\Pr(X < b/3)$, we set $t = \frac{b-3}{3\sqrt{b-1}}$ to get

$$\Pr(X < b/3) \le e^{-\frac{(b-3)^2}{18(b-1)}} \le e^{-b/19} \quad \text{for } b \ge 95,$$

where the condition $b \ge 95$ comes from rounding the solution to a quadratic equation. □

Lemma 3. *In $\mathcal{P}'$, Conversion to Silence occurs with probability at most $\delta/3$.*

Proof. Recall that in round r, each encoded message includes $38\left\lceil \log\left(\frac{2n\pi r}{\sqrt{\delta}}\right)\right\rceil$ random bits at the end. Then, Lemma 2 tells us that the probability that adversary is able to flip these random bits of a word to forge silence is at most $2^{-\frac{38\left\lceil \log\left(\frac{2n\pi r}{\sqrt{\delta}}\right)\right\rceil}{19}} \le (n\pi r)^{-2}\,\delta/4$. Thus, similar to Lemma 1, for $1 \le k \le 4, 1 \le i \ne j \le n$ and $r \ge 1$, define $\xi_{i,j,k,r}$ to be the event that in round r, the k^{th} word exchanged between players i and j is converted to silence, so that $\Pr(\xi_{i,j,k,r}) \le (n\pi r)^{-2}\,\delta/4$. Hence, by using a similar union bound as Lemma 1, we get the desired bound. □

Lemma 4. *In $\mathcal{P}'$, the adversary is able to guess the key of some player with probability at most $\delta/3$.*

Proof. Recall that the keys are of length $2\left\lceil \log\left(\frac{4n\pi r}{\sqrt{\delta}}\right)\right\rceil$ in round r. Each such key is generated uniformly at random from the set of all binary strings of this length. Thus, the probability of guessing this key is at most $2^{-2\left\lceil \log\left(\frac{4n\pi r}{\sqrt{\delta}}\right)\right\rceil} \le (n\pi r)^{-2}\,\delta/16$. Again, similar to Lemma 1, for $1 \le k \le 4, 1 \le i \ne j \le n$ and $r \ge 1$, define $\xi_{i,j,k,r}$ to be the event that in round r, the adversary is able to guess the key in the k^{th} word exchanged between players i and j, so that $\Pr(\xi_{i,j,k,r}) \le (n\pi r)^{-2}\,\delta/16$. Hence, by using a similar union bound as Lemma 1, we get the desired bound. □

Lemma 5. *With probability at least $1-\delta$, none of the failure events happen during a run of $\mathcal{P}'$.*

Proof. A run of protocol $\mathcal{P}'$ fails if any of the three failure events described above happen. From Lemmas 1, 3 and 4, the total probability of failure is computed by using union bound over the three failure events, which gives $\delta/3+\delta/3+\delta/3 = \delta$. Hence, the run succeeds with probability at least $1-\delta$. □

Next, now show that $\mathcal{P}'$ correctly simulates all valid runs of $\mathcal{P}$.

B. Correctness

All lemmas in this section assume that none of the failure events occur. Without loss of generality, we also assume that Alice and Bob generate their keys in every round. We use the phrase *terminated in* $\mathcal{P}$ to mean finished executing line 13 of Algorithm 3.

Lemma 6. *Fix a round r of $\mathcal{P}'$. Let W be any of the four sequences of w_r bits on the channel from Alice to Bob or the channel from Bob to Alice in this round and $k_A^{(r)}$, $k_B^{(r)}$ be the keys that Alice and Bob generate in this round. Then exactly one of the following hold.*

1) *$\texttt{IsSilence}(W)$ is true, in which case the sender on the channel was silent as well.*
2) *$\mathscr{D}_r(W) = \mathcal{C} \neq \perp$. Then,*
    a) *If $\mathcal{C} = (\texttt{KEY?}, k)$, then either Alice sent W with key $k = k_A^{(r)}$, or Alice is silent and the adversary sent W.*
    b) *If $\mathcal{C} = (k', k_A^{(r)})$, then Bob has not* terminated in *$\mathcal{P}$ yet, and he sent W with key $k' = k_B^{(r)}$.*
    c) *If $\mathcal{C} = (x, k_B^{(r)})$, then Alice sent W with content $x = m^{(r)}$, where $m^{(r)}$ is Alice's message for Bob in round r.*
    d) *Otherwise, the adversary sent W on a silent channel.*
3) *Bob executed line 18 of Algorithm 2.*
4) *W is the outcome of adversarial tampering on the channel.*

Proof. In any given round r, whenever Alice or Bob send a word to the other, this word does not convert to silence because we assume that the failure event *Conversion to Silence* does not occur. Hence, if silence is received on the channel, then the sender on the channel must have been silent at that time. This proves part (1) of our lemma.

If, however, a valid AMD codeword is received, which decodes into $\mathcal{C}$, the following cases are possible. Case 2(a) : If $\mathcal{C} = (\texttt{KEY?}, k)$, then either Alice issued this key request to Bob with her key $k_A^{(r)}$ (line 5 of Algorithm 1), or she was silent. In the former case, the adversary is unable to put a key $k \neq k_A^{(r)}$ in the codeword since the failure event *AMD Failure* does not occur, and hence, $k = k_A^{(r)}$. In the latter case, since the channel is silent, the adversary must have issued the key request with some key k. Case 2(b) : If $\mathcal{C} = (k', k_A^{(r)})$, then since we assume that the failure event *Key Installation* does not occur, it must be the case that Bob sent W and not the adversary. Thus, Bob must have included his own key $k_B^{(r)}$ to the codeword along with a copy of Alice's key (line 14 of Algoithm 2). This is only possible when Bob has not *terminated in* $\mathcal{P}$ yet. Case 2(c) : If $\mathcal{C} = (x, k_B^{(r)})$, then Alice must have sent W since the the failure event *Key Installation* does not occur. Also, since we assume that the failure event *AMD Failure* does not occur, the adversary would have not been able to convert $m^{(r)}$ into another message successfully, and hence, W must contain the message that Alice has for Bob in this round. Case 2(d) : If the AMD codeword is neither of the above three cases, then it must be the case that the sender was on the channel at the time W was on the channel, since *AMD Failure* does not occur. Hence, the adversary must have sent W on a silent channel.

If neither silence nor an AMD codeword is received, then W is noise. The only step in $\mathcal{P}'$ where noise is intentionally put on the channel is when Bob has to inform Alice that he did not receive her message correctly (line 18 of Algorithm 2). Thus, if Bob sent this noise, Case 3 of our lemma holds, else Case 4 must hold where the adversary has tampered with the bits on the channel so that W becomes noise. □

Lemma 7. *Assume Alice calls SEND-MSG(m) in some round r_1 for some message m and bit b. Then the following hold: (1) Alice returns from SEND-MSG(m) in some round $r_3 \geq r_1$; and (2) either Bob records the message m (line 22 of Algorithm 2) in exactly one round r_2 where $r_1 \leq r_2 \leq r_3$, or Bob does not record the message m between rounds r_1 and r_3 because Bob* terminated in $\mathcal{P}$.

Proof. We first show that if Alice calls SEND-MSG(m) in round r_1, then there must exist some round $r_3 \geq r_1$ in which this call returns. Since the adversary's budget is finite, there must be some round, r_4, after which no bits are ever flipped. If the call returns before round r_4, then part (1) is proven, so we only consider the cases where the call extends past round r_4. In round $r_4 + 1$, after Alice has sent a key request she either hears silence, indicating that Bob has *terminated in* $\mathcal{P}$, and the call returns, or she correctly receives Bob's key, and uses it to send m which Bob correctly receives, and records (since the bit b is different than the bit in the last message recorded). Next, Alice hears silence from Bob. This ends the call. Either way the call returns at the end of round $r_4 + 1$. Thus, in every case there is some round $r_3 \geq r_1$ in which the call to SEND-MSG(m) returns.

We now prove part (2) of our lemma. We first show that the message m is recorded at most once by Bob in the rounds r_1 to r_3. By Lemma 6 (2(c)), the bit b' received by Bob in line 20 of Algorithm 2 must be the same as the bit b sent by Alice from rounds r_1 to r_3. Since this bit never changes, Bob will record a message at most once in rounds r_1 to r_3.

If Bob *terminated in* $\mathcal{P}$ before round r_3, then part (2) of our lemma statement does not require Bob to record the message m, and so that part or our lemma is trivially true. Thus, for the remainder of the proof, we assume that Bob has not *terminated in* $\mathcal{P}$ before round r_3.

Consider round r_3 in which the call to SEND-MSG(m) returns. Let M_1 be the string read by Alice on line 6 of Algorithm 1. Since Bob has not terminated, Lemma 6 (2(b)) guarantees that Bob sent M_1, and therefore the call to Algorithm 2 does not return on line 8. Hence, it returns on line 18. Since this is the last round, M_1 must correctly decode to (k_B, k_A) in Algorithm 1. Thus, Alice sends Bob m using k_B and hears silence subsequently. Thus, Bob must have actually been silent at this time (Lemma 6 (1)), which only happens if he has either now or previously recorded m. Hence, Bob must have recorded the message m in some round $r_2 \leq r_3$. □

Lemma 8. *The following holds for $\mathcal{P}'$. For any message m that Bob records (line 22 of Algorithm 2) in some round r_2, Alice started a call to SEND-MSG with the message m, in a round $r_1 \leq r_2$ and returned from that call in some round $r_3 \geq r_2$.*

Proof. Consider the round r_2 in which the Bob records a message m (line 22 of Algorithm 2). In round r_2, in line 20 of Algorithm 2, let $\mathscr{D}_r\left(M_1'\right) = ((m,b),k_B)$. Thus, by Lemma 6 (2(c)), Alice must have sent $\mathscr{E}_r\left((m,b),k_B\right)$, during the transmission of M_1'. Hence, Alice must be in a call to SEND-MSG with the message m, and this call must have begun in some round $r_1 \leq r_2$.

Finally, we know by Lemma 7(1) that every call to SEND-MSG ends in some round r_3. Since, by the above, Alice is in the call during round r_2, it must be the case that Alice returns from the call in some round $r_3 \geq r_2$. □

Lemma 9. *Algorithm $\mathcal{P}'$ terminates with a correct simulation of an asynchronous run of $\mathcal{P}$.*

Proof. We show that for every pair of players u and v, protocol $\mathcal{P}'$ correctly simulates a FIFO message channel(see [31] Chapter 14.1.2) from u to v during the simulation of $\mathcal{P}$. Then a direct induction shows that for each player u, $\mathcal{P}'_u$ simulates $\mathcal{P}_u$ correctly.

Fix an arbitrary channel from u to v in the network. Let $Q_{u,v}$ be the queue of SEND-MSG procedures that are maintained in $\mathcal{P}'$ by u of messages to send to v.

We require the following facts:

1) In any time step, there is a transition in $\mathcal{P}_u$ across a transition relation with an output action to send a message m to player v, if and only if the procedure SEND-MSG for message m to player v is pushed on the queue $Q_{u,v}$ in that time step.
2) Every SEND-MSG procedure for message m on $Q_{u,v}$ will eventually start at some round r_1 and end in some round r_3. Moreover, player v transitions across an input transition relation for message m from player u at most once in some round r_2, $r_1 \leq r_2 \leq r_3$. Moreover, if there is no such input transition relation, than $\mathcal{P}_u$ has entered a termination state before round r_3.
3) For any transition along a transition relation in $\mathcal{P}_v$ with an input action, in some round r_2, player u started a call to SEND-MSG with the message m, in a round $r_1 \leq r_2$ and returned from that call in some round $r_3 \geq r_2$.

Fact (1) follows directly from Algorithm 3 steps 3-6. The first sentence of Fact (2) follows by induction and Lemma 7, and the remainder of the fact follows directly from Lemma 7. Fact (3) follows from Lemma 8.

Together, the facts show that no matter what the actions of the adversary, the protocol $\mathcal{P}'$ correctly simulates a FIFO message channel from player u to player v. In particular, we have: 1) when a transition is taken in $\mathcal{P}_u$ with output action to send message m to player v, this message is put on a queue; 2) all transitions in $\mathcal{P}_v$ with an input action to receive a message m from player u are associated with the removal of message m from the queue; and 3) all messages are eventually removed from the queue, triggering transitions across transition relations with input actions in $\mathcal{P}_v$, unless $\mathcal{P}_v$ is already in a termination state. □

C. Resource Costs

We now compute the expected number of bits sent and the latency of $\mathcal{P}'$.

Lemma 10. *In $\mathcal{P}'$, round $r \geq 1$ begins at time step $\tau(r) = \Theta\left(r \log\left(\frac{nr}{\delta}\right)\right)$.*

Proof. For all $r \geq 1$, note that round $r+1$ begins as soon as the number of time steps corresponding to four words of the round r have passed. Hence, we can compute $\tau(r)$ using the recurrence $\tau(r) = \tau(r-1) + 4w_{r-1}$, where $\tau(1) = 1$. This gives $\tau(r) = \tau(1) + 4\sum_{i=1}^{r-1} w_i$. Now, since $w_r = O(\log(nr/\delta))$ (as discussed in Section III-B), using $w_i \leq C_1 \log\left(\frac{ni}{\delta}\right) + C_2$, we get $\tau(r) \leq 1 + 4\sum_{i=1}^{r-1}\left(C_1 \log\left(\frac{ni}{\delta}\right) + C_2\right)$. Finally, using the fact that $\sum_{i=1}^{r} \log i = \Theta(r \log r)$, we get $\tau(r) = \Theta\left(r \log\left(\frac{nr}{\delta}\right)\right)$. □

Lemma 11. *If $\tau(x) \leq z$ for some $x \geq 1$ and $z \geq 1$, then $x = O(z/\log\left(\frac{nz}{\delta}\right))$.*

Proof. We prove the bound on x for the case that $\tau(x) = z$. By Lemma 10, we know that $\tau(x) = C\left(x \log\left(\frac{nx}{\delta}\right)\right)$, for some constant C. Thus, we have $z = Cx \log\left(\frac{nx}{\delta}\right)$, and we get (*) $x \leq z/(C \log\left(\frac{nx}{\delta}\right))$.

Note that $z = Cx \log\left(\frac{nx}{\delta}\right) \leq Cx \log x$. Taking logs of both sides, we get that $\log z \leq C' \log x$ for some constant C', which implies that $\log\left(\frac{nz}{\delta}\right) \leq C' \log\left(\frac{nx}{\delta}\right)$. Now plugging this back into (*), we get that $x \leq C''z/\log\left(\frac{nz}{\delta}\right)$ for some constant C''. □

Lemma 12. *If $\mathcal{P}'$ succeeds, then it has the following resource costs.*

- *The number of bits sent is $O\left(L \log\left(\frac{nL}{\delta}\right) + T\right)$.*
- *The latency is $O\left(\max_p \left\{\Lambda_p \log\left(\frac{n(L+T)}{\delta}\right) + T_p\right\}\right)$, where the maximum is taken over all communication paths, p, in the asynchronous simulation of $\mathcal{P}$, Λ_p is the latency of p, and T_p is the total number of bits flipped by the adversary on edges in p.*

Proof. To bound the number of bits sent, we assume pessimistically that in every round of $\mathcal{P}'$, there is an attempt to send exactly one message. This maximizes the number of bits sent since word sizes increase with time.

Since each word in $\mathcal{P}'$ is ECC encoded, the adversary must flip a constant fraction of the bits to successfully corrupt the word, and thereby compromise the round. Let x be the number of rounds in which some word was successfully corrupted by the adversary. Since the length of the words increases with successive rounds, T must at least be a constant, C, times the number of bits in the first x rounds of $\mathcal{P}'$, and hence, we must have $\tau(x) \leq (T+1)/C$.

By Lemma 11, we know that $x = O\left(\frac{T}{\log(n(T+1)/\delta)}\right)$. Thus, since $\mathcal{P}'$ requires L progressive rounds, the total number of rounds is $r = L + O\left(\frac{T}{\log(n(T+1)/\delta)}\right)$.

Hence the total number of bits sent is at most $\tau(r)$. By Lemma 10,

$$\begin{aligned}
\tau(r) &= O\Bigg(\left(L + \frac{T}{\log\left(n(T+1)/\delta\right)}\right) \\
&\qquad \log\left(\frac{n}{\delta}\left(L + \frac{T}{\log\left(n(T+1)/\delta\right)}\right)\right)\Bigg) \\
&\leq O\left(\left(L + \frac{T}{\log\left(n(T+1)/\delta\right)}\right)\log\frac{n(L+T)}{\delta}\right) \\
&= O\left(L\log\left(\frac{n(L+T)}{\delta}\right) + T\right) \\
&= O\left(L\log\left(\frac{nL}{\delta}\right) + T\right).
\end{aligned}$$

The second line above follows from $\frac{T}{\log(n(T+1)/\delta)} \leq T$, and the third line from the fact that if $L = O(T)$, then $\log(L+T) = O(\log T)$. The final line above follows from the fact that $\log(L+T) = \log(L) + \log(1+T/L) \leq \log(L) + T/L$, and hence $L\log(L+T) \leq L\log(L) + T$. This bounds the number of bits sent.

To bound the latency, we note that the argument above holds for any communication path p in the asynchronous simulation of $\mathcal{P}$. For the p which achieves the maximum, it follows by induction that all other required messages will have already been received by the time they are needed, and so p determines the overall latency. □

We are finally ready to complete the proof of Theorem 1.

Proof of Theorem 1 with $\alpha = 1$. By Lemma 5, no failure event happens with probability at least $1-\delta$, and by Lemma 9, in such a situation, $\mathcal{P}'$ terminates with an asynchronous simulation of $\mathcal{P}$. Upon correct termination, the resource cost bounds hold by Lemma 12. □

Appendix B
Simulating Protocols with Messages of Arbitrary Length

We now show how to reduce resource costs when the average message length in $\mathcal{P}$ is high, even if this average is unknown *a priori*. For this section, we first change $\mathcal{P}$ so that all messages of $\mathcal{P}$ are from a prefix-free language, and so it is possible to detect when a message of $\mathcal{P}$ ends. This can be done with at most constant bit blowup [13].

A. Algorithm

The main simulating algorithm $\mathcal{P}'$ (Algorithm 3) remains unchanged in this setting. Only the sending and receiving algorithms need to change to reflect the fact that the message to be sent may be longer than a single call to $\mathscr{E}_r(,)$ can support. In Algorithms 4 and 5 below, we highlight the necessary changes in red. For any given round r, we denote by WORD-PARAMS(r) the function which returns a tuple (w_r, κ_r) where w_r is the word length in that round and κ_r is the key length in this round. These values are computed in the same way as the previous case. We analyze Algorithms 4 and 5 in Section C.

Algorithm 4 Message exchange algorithm for the sender.

```
1:  procedure SEND-MSG($m$)
2:      ▷ $b$ is a persistent variable for the parity bit. On the
        first call to SEND-MSG, $b$ is set to 0. On subsequent calls
        it is whatever it was set to on the previous call.
3:      $j \leftarrow 0$
4:      while $j < |m|$
5:          $(w_r, \kappa_r) \leftarrow$ WORD-PARAMS($r$)
6:          Generate random key $k_A$ of length $\kappa_t$.
7:          Send $\mathscr{E}_r$ (KEY?, $k_A$).
8:          $M_1 \leftarrow w_r$ bits from the receiver.
9:          if IsSilence ($M_1$) then
10:             ▷ Assume the receiver has already terminated.
11:             $b \leftarrow \neg b$
12:             Stay silent for $2w_r$ time steps and return
13:         else
14:             $(x, k) \leftarrow \mathscr{D}_r(M_1)$
15:             if $k \neq k_A$ then
16:                 Stay silent for $2w_r$ time steps.
17:             else
18:                 $k_B \leftarrow x$
19:                 $M \leftarrow m[j, j+\kappa_t]$     ▷ Next $\kappa_t$ bits of $m$
20:                 Send $\mathscr{E}_r((M, b), k_B)$.
21:                 $M_2 \leftarrow w_r$ bits from the receiver.
22:                 if IsSilence ($M_2$) then
23:                     $j \leftarrow j + \kappa_t$
24:                     $b \leftarrow \neg b$
25:         return
```

Appendix C
Analysis of Algorithm 4 and 5

A. Correctness

Lemma 13. *Assume Alice calls SEND-MSG(m) in some round r_1 for some message m and bit b. Then the following hold: (1) Alice returns from SEND-MSG(m) in some round $r_3 \geq r_1$; and (2) either Bob records the message m (line 34 of Algorithm 5) in exactly one round r_2 where $r_1 \leq r_2 \leq r_3$, or Bob does not record the message m between rounds r_1 and r_3 because he* terminated in $\mathcal{P}$.

Proof. We first show that if Alice calls SEND-MSG(m) in round r_1, then there must exist some round $r_3 \geq r_1$ in which this call returns. Since the adversary's budget is finite, there

Algorithm 5 Message exchange algorithm for the receiver.

```
1:  procedure RECEIVE-MSG( )
2:      ▷ $\hat{b}, \mu, \lambda$ are persistent variables for the parity bit, the partially received message, and length of the recorded message, respectively. On the first call to RECEIVE-MSG, $\hat{b} \leftarrow 0$, $\mu \leftarrow \varnothing$, $\lambda \leftarrow 0$. On subsequent calls these variables are whatever they were set to on the previous call.
3:      $(w_r, \kappa_r) \leftarrow$ WORD-PARAMS$(r)$
4:      $M_1' \leftarrow w_r$ bits on the channel from the sender.
5:      if IsSilence$(M_1')$ then
6:          Stay silent for $3w_r$ time steps.
7:      else
8:          $(x', k') \leftarrow \mathscr{D}_r(M_1')$
9:          if $x' \neq$ KEY? then
10:             Send noise for $w_r$ time steps.
11:             Stay silent for $2w_r$ time steps.
12:         else
13:             $k_A \leftarrow k'$.
14:             Generate random key $k_B$ of length $\kappa_t$.
15:             Send $\mathscr{E}_r(k_B, k_A)$.
16:             $M_2' \leftarrow w_r$ bits on the channel from the sender.
17:             $(x'', k'') \leftarrow \mathscr{D}_r(M_2')$
18:             if $k'' \neq k_B$ then
19:                 Send noise for $w_r$ time steps.
20:             else
21:                 $(M', b') \leftarrow x''$
22:                 if $\mu = \varnothing$ then
23:                     if $b' \neq \hat{b}$ then
24:                         $\lambda \leftarrow |M'|$
25:                         $\mu \leftarrow M'$
26:                 else
27:                     if $b' \neq \hat{b}$ then
28:                         $\lambda \leftarrow |M'|$
29:                         Append $M'$ to $\mu$.
30:                     else
31:                         Replace last $\lambda$ bits of $\mu$ with $M'$.
32:                         $\lambda \leftarrow |M'|$ do
33:                 if $\mu$ is a completed message of $\mathscr{L}$ then
34:                     Record the message $\mu$ from the sender.
35:                     $\mu \leftarrow \varnothing$
36:                     $\hat{b} \leftarrow b'$
37:                 Stay silent for $w_r$ time steps.
```

must be some round, r_4, after which no bits are ever flipped. If the call returns before or during round r_4, then part (1) is proven, so we only consider the cases where the call extends past round r_4. Let m' be the part of the message that remains to be sent. For $i \geq 1$ consider round $r_4 + i$. There are two possibilities:

(a) After Alice has sent a key request in round $r_4 + i$, she hears silence and the call returns, or;
(b) Alice correctly receives Bob's key, and uses it to send the next piece of m which Bob correctly receives. Next, Alice hears silence from Bob. If this was the last piece of the message, this ends the call. If not, the call continues into round $r_4 + i + 1$ with a shorter remaining message.

Since the message has finite length, eventually (a) occurs. Thus, in every case there is some round $r_3 \geq r_1$ in which the call to SEND-MSG(m) returns.

We now prove part (2) of our lemma. We first show that the message m is recorded at most once by Bob in the rounds r_1 to r_3. By Lemma 6 (2(c)), the bit b' corresponding to the last partial message for m received by Bob in line 21 of Algorithm 5 must be the same as the bit b sent by Alice for this partial message from rounds r_1 to r_3. Since this bit never changes, Bob will record m (line 34 of Algorithm 5) at most once in rounds r_1 to r_3.

If Bob *terminated in* $\mathcal{P}$ before round r_3, then part (2) of our lemma statement does not require Bob to record the message m, and so that part or our lemma is trivially true. Thus, for the remainder of the proof, we assume that Bob has not *terminated in* $\mathcal{P}$ before round r_3 and we must show that he records the message m exactly once.

We do this by induction on the length of m. Note that since m is a message from $\mathcal{P}$, it belonged to $\mathscr{L}$, so it is actually possible for Bob to record m. If m is short enough to be sent in one piece, then consider round r_3 in which the call to SEND-MSG(m) returns. Let M_1 be the string read by Alice on line 8 of Algorithm 4. Since Bob has not *terminated in* $\mathcal{P}$, Lemma 6 (2(b)) guarantees that Bob sent M_1, and therefore the call to Algorithm 4 does not return on line 8. Hence, it returns on line 25. Since this is the last round, M_1 must correctly decode to (k_B, k_A) in Algorithm 4. Thus, Alice sends Bob m using k_B and hears silence subsequently. Thus, Bob must have actually been silent at this time (Lemma 6 (1)), which only happens if he has either now or previously recorded m. Hence, Bob must have recorded the message m in some round $r_2 \leq r_3$.

Now as an induction hypothesis suppose the conclusion about Bob recording a message exactly once is true for all strings s shorter than m. Suppose m requires more than one piece to be sent. Then since Alice continues to resend the first piece m_0 until she has received confirmation that Bob has received it, there is some round $r_1' < r_3$ in which Alice receives this confirmation. Since the same parity bit b for this piece has been sent in all rounds $r \leq r_1'$, and received bit b' agrees with b in each of these rounds, Bob stores the partial message m_0 exactly once. Let m_1 be the remaining portion of m. Clearly it is shorter than m. Now let us examine the control flow throughout the algorithm from round r_1' onwards. This looks exactly like a call to SEND-MSG(m_1) with the persistent variable b now set to $\neg b$, together with a RECEIVE-MSG call in which the persistent variables $\hat{b}$ set to $\neg\hat{b}$, μ set to m_0, and λ set to $|m_0|$. By induction hypothesis, there is exactly one round r_2 with $r_1' \leq r_2 \leq r_3$ during which Bob records $m_0 \circ m_1$, which is in $\mathscr{L}$. But since $m = m_0 \circ m_1$, this concludes the proof. □

Lemma 14. *The following holds for protocol $\mathcal{P}'$. For any message m that Bob records (line 34 of Algorithm 5) in some round r_2, Alice started a call to SEND-MSG(m) in a round $r_1 \leq r_2$ and returned from that call in some round $r_3 \geq r_2$.*

Proof. Consider the round r_2 in which the Bob records a message m (line 34 of Algorithm 5). Suppose $\mu = \varnothing$ when RECEIVE-MSG was called in round r_2. Then m arose in the first component of $\mathscr{D}_r\left(M_1'\right)$ in line 21 of Algorithm 5, where the second component correctly matched k_B. Specifically the first component must have been (m, b) for some b. Thus, by Lemma 6 (2(c)), Alice must have sent $\mathscr{E}_r\left((m, b), k_B\right)$, during the transmission of M_1'. Hence, Alice must be in a call to SEND-MSG with some message $\hat{m}$ from $\mathscr{L}$, with m as a contiguous substring, and this call must have begun in some round $r_1 \leq r_2$. We must show that $\hat{m} = m$. Suppose m is a proper contiguous substring of $\hat{m}$, *i.e.*, $\hat{m} = m_0 \circ m \circ m_1$ where at least one of m_0 and m_1 is not $\varnothing$. Then m_0 is not in $\mathscr{L}$ (since it is a prefix of $\hat{m}$). Since Alice would not proceed to sending m until she had confirmed that Bob had received m_0, she must have previously received that confirmation, meaning that Bob previously received m_0. But in that case, Bob would have that stored as a partial message, contradicting the assumption that $\mu = \varnothing$ when RECEIVE-MSG was called in round r_2. Thus $m_0 = \varnothing$. But that means that m is a prefix of $\hat{m}$. Since both m and $\hat{m}$ are in $\mathscr{L}$, it follows that $m_1 = \varnothing$ and $\hat{m} = m$. Thus, Alice called SEND-MSG(m).

Next suppose $\mu = s$ when RECEIVE-MSG was called in round r_2, for some string $s \neq \varnothing$. Then $m = s \circ x$, where x is the string Bob decodes in round r_2. We first induct on the length of s to prove that Alice's call to SEND-MSG had as input, some message $\hat{m}$ from $\mathscr{L}$, such that $\hat{m}$ contains $s \circ x = m$ as a contiguous substring, *i.e.*, $\hat{m} = m_0 \circ m \circ m_1$ for some strings m_0, m_1. The case of $s = \varnothing$ was shown above. Thus, assume there exist rounds $\rho_0 < \rho_1 \leq r_2$ and non-empty strings s_0 and s_1, with $s = s_0 \circ s_1$ such that $\mu = s_0$ when RECEIVE-MSG was called in round ρ_0 and $\mu = s_1$ when RECEIVE-MSG was called in round ρ_1. Here, ρ_0 and ρ_1 are the first rounds with this property. Then Bob decoded s_0 in round $\rho_0 - 1$ and s_1 in round $\rho_1 - 1$. Furthermore, Bob has not decoded any other string between decoding s_0 and s_1. Then, it follows from the induction hypothesis that s_0 and s_1 came from Alice, so that Alice must be in call to SEND-MSG during the course of which she sends encoded strings s_0, s_1, x in this order. Moreover, she cannot have sent any other encoded strings in between, because had she done so, she would not have moved on from it without acknowledgement that Bob decoded it. Since we stipulate that Bob did not decode any other strings, it follows that the input to Alice's call to SEND-MSG had $m = s_0 \circ s_1 \circ x$ as a contiguous substring so that $\hat{m} = m_0 \circ m \circ m_1$ for some strings m_0, m_1. Note that m_0 is not in $\mathscr{L}$. Now, when Bob decoded s_0, the call to RECEIVE-MSG was passed $\varnothing$. Thus, from the discussion in the beginning of the proof, it follows that $m_0 = \varnothing$ and therefore $m_1 = \varnothing$ and $\hat{m} = m$. Thus, in all cases, Alice did actually have a call to SEND-MSG(m) that began in some round $r_1 \leq r_2$.

Finally, we know by Lemma 13(1) that every call to SEND-MSG ends in some round r_3. Since, by the above, Alice is in the call during round r_2, it must be the case that Alice returns from the call in some round $r_3 \geq r_2$. □

B. Resource Costs

We now analyze the total number of bits sent by $\mathcal{P}'$ when the messages in $\mathcal{P}$ may be longer than a single bit. Let α be the average message length in $\mathcal{P}$.

Lemma 15. *If $\mathcal{P}'$ succeeds, then it has the following resource costs.*

- *The number of bits sent is $O\left(L\left(1 + \frac{1}{\alpha}\log\left(\frac{nL}{\delta}\right) + T\right)\right)$.*
- *The latency is $O\left(\max_p\left\{\Lambda_p\left(1 + \frac{1}{\alpha}\log\left(\frac{n(L+T)}{\delta}\right) + T_p\right\}\right)$ where p is any communication path in the asynchronous simulation of $\mathcal{P}$, Λ_p is the latency of p, and T_p is the total number of bits flipped by the adversary on edges in p.*

Proof. Call a string M encoded in Step 20 of Algorithm 4 a *submessage*. To upper-bound the number of bits sent, we assume pessimistically that in every round of $\mathcal{P}'$, there is an attempt to send exactly one submessage. This maximizes the number of bits sent since word sizes increase with time.

Note that each message consists of at most 1 submessage that is not of length some constant times the word length in that round. Thus, the number of progressive rounds is no more than the number of messages sent, L/α, plus the largest integer x such that the number of bits sent in x rounds equals $c_1 L$ for some constant $c_1 > 1$. By Lemma 11, $x = O(L/\log\frac{nL}{\delta})$.

The number of non-progressive rounds is $O\left(\frac{T}{\log(n(T+1)/\delta)}\right)$, by the same argument as from the proof of Lemma 12.

Let the number of rounds

$$r = L/\alpha + O(L/\log\frac{nL}{\delta}) + O\left(\frac{T}{\log\left(n(T+1)/\delta\right)}\right).$$

Then by Lemma 10, we can bound $\tau(r)$ as

$$\tau(r) \leq \left(L/\alpha + O\left(\frac{L}{\log(nL/\delta)}\right) + z\right)\left(\log\frac{nr}{\delta}\right)$$

where $z = O\left(\frac{T}{\log(n(T+1)/\delta)}\right)$. Simplifying this, we get $\tau(r) = O\left(\frac{1}{\alpha}\log\left(\frac{n(L+T)}{\delta}\right) + L + T\right)$.

Finally, to bound the latency, we note that the argument above holds for any communication path p in the asynchronous simulation of $\mathcal{P}$.

□

Proof of Theorem 1. By Lemma 5, no failure event happens with probability at least $1 - \delta$, and by Lemma 9, in such a situation, $\mathcal{P}'$ terminates with an asynchronous simulation of $\mathcal{P}$. Upon correct termination, the resource cost bounds hold by Lemma 15. □