Bridging the Capacity Gap Between Interactive and One-Way Communication

Bernhard Haeupler[*]
Carnegie Mellon University
haeupler@cs.cmu.edu

Ameya Velingker [†]
Carnegie Mellon University
avelingk@cs.cmu.edu

Abstract
We study the communication rate of coding schemes for interactive communication that transform any two-party interactive protocol into a protocol that is robust to noise.

Recently, Haeupler [11] showed that if an $\epsilon > 0$ fraction of transmissions are corrupted, adversarially or randomly, then it is possible to achieve a communication rate of $1 - \widetilde{O}(\sqrt{\epsilon})$. Furthermore, Haeupler conjectured that this rate is optimal for general input protocols. This stands in contrast to the classical setting of one-way communication in which error-correcting codes are known to achieve an optimal communication rate of $1 - \Theta(H(\epsilon)) = 1 - \widetilde{\Theta}(\epsilon)$.

In this work, we show that the quadratically smaller rate loss of the one-way setting can also be achieved in interactive coding schemes for a very natural class of input protocols. We introduce the notion of *average message length*, or the average number of bits a party sends before receiving a reply, as a natural parameter for measuring the level of interactivity in a protocol. Moreover, we show that any protocol with average message length $\ell = \Omega(\text{poly}(1/\epsilon))$ can be simulated by a protocol with optimal communication rate $1 - \Theta(H(\epsilon))$ over an oblivious adversarial channel with error fraction ϵ. Furthermore, under the additional assumption of access to public shared randomness, the optimal communication rate is achieved *ratelessly*, i.e., the communication rate adapts automatically to the actual error rate ϵ without having to specify it in advance.

This shows that the capacity gap between one-way and interactive communication can be bridged even for very small (constant in ϵ) average message lengths, which are likely to be found in many applications.

1 Introduction

In this work, we study the communication rate of coding schemes for interactive communication that transform any two-party interactive protocol into a protocol that is robust to noise.

1.1 Error-Correcting Codes The study of reliable transmission over a noisy channel was pioneered by Shannon's work in the 1940s. He and others showed that error-correcting codes allow one to add redundancy to a message, thereby transforming the message into a longer sequence of symbols, such that one can recover the original message even if some errors occur. This allows fault-tolerant transmissions and storage of information. Error-correcting codes have since permeated most modern computation and communication technologies.networks, data storage, computation, or data computation, dainvolving data storage, transmission or csuch as storage devices (e.g., disk drives, RAM, DVDs), satellite systems, wireless LAN, etc.

One focus of study has been the precise tradeoff between redundancy and fault-tolerance. In particular, if one uses an error-correcting code that encodes a binary message of length k into a sequence of n bits, then the *communication rate* of the code is said to be k/n. One wishes to make the rate as high as possible. Shannon showed that for the random binary symmetric channel (BSC) with error probability ϵ the (asymptotically) best achievable rate is $C = 1 - H(\epsilon)$, where $H(\epsilon) = -\epsilon \log_2 \epsilon - (1-\epsilon)\log_2(1-\epsilon)$ denotes the *binary entropy function*.

Another realm of interest is the case of *adversarial* errors. In this case, the communication channel corrupts at most an ϵ fraction of the total number of bits that are transmitted. Moreover, one wishes to allow the receiver to correctly decode the message in the presence of any such error pattern. The work of Hamming shows that one can achieve a communication rate of $R = 1 - \Theta(H(\epsilon))$, in particular, the so-called Gilbert-Varshamov bound of $1 - H(2\epsilon) > 1 - 2H(\epsilon)$. Finding the optimal rate, or even just the constant in the asymptotic $\Theta(H(\epsilon))$ term, remains a major open question.

1.2 Interactive Communication The work of Shannon and Hamming applies to the problem of *one-way communication*, in which one party, say Alice,

[*]Computer Science Department, Carnegie Mellon University. Research supported in part by NSF grant CCF-1527110 and the NSF-BSF grant "Coding for Distributed Computing."

[†]Computer Science Department, Carnegie Mellon University. Part of this work was done while the author was visiting the Simons Institute for the Theory of Computing, Berkeley, CA. Research supported in part by NSF grant CCF-0963975.

wishes to send a message to another party, say Bob. However, in many applications, underlying (two-party) communications are *interactive*, i.e., Bob's response to Alice may be based on what he received from her previously and vice versa. As in the case of one-way communication, one wishes to make such interactive communications robust to noise by adding some redundancy.

At first sight, it seems plausible that one could use error-correcting codes to encode each round of communication separately. However, this does not work correctly because the channel might corrupt the codeword of one such round of communication entirely and as a result derail the entire future conversation. With the naive approach being insufficient, it is not obvious whether it is possible at all to encode interactive protocols in a way that can tolerate some small constant fraction of errors in an interactive setting. Nonetheless, Schulman [14, 15, 16] showed that this is possible and numerous follow-up works over the past several years have led to a drastically better understanding of error-correcting coding schemes for interactive communications.

1.3 Communication Rates of Interactive Coding Schemes Only recently, however, has this study led to results shedding light on the tradeoff between the achievable communication rate for a given error fraction or amount of noise.

Kol and Raz [12] gave a communication scheme for random errors that achieves a communication rate of $1-O(\sqrt{H(\epsilon)})$ for any alternating protocol, where $\epsilon > 0$ is the error rate. [12] also developed powerful tools to prove upper bounds on the communication rate. Haeupler [11] showed communication schemes that achieve a communication rate of $1 - O(\sqrt{\epsilon})$ for any oblivious adversarial channel, including random errors, as well as a communication rate of $1 - O(\sqrt{\epsilon \log\log(1/\epsilon)})$ for any fully adaptive adversarial channel. These results apply to alternating protocols as well as adaptively simulated non-alternating protocols (see [11] for a more detailed discussions). Lastly, given [12], Haeupler conjectured these rates to be optimal for their respective settings. Therefore, there is an almost quadratic gap between the conjectured rate achievable in the interactive setting and the $1 - \Theta(H(\epsilon))$ rate known to be optimal for one-way communications.

1.4 Results In this paper, we investigate this communication rate gap. In particular, we show that for a natural and large class of protocols this gap disappears. Our primary focus is on protocols for *oblivious adversarial* channels. Such a channel can corrupt any ϵ fraction of bits that are exchanged in the execution of a protocol, and the simulation is required to work, with high probability, for any such error pattern. This is significantly stronger, more interesting, and, as we will see, also much more challenging than the case of independent random errors. We remark that, in contrast to a *fully adaptive adversarial* channel, the decision whether an error happens in a given round is not allowed to depend on the transcript of the execution thus far. This seems to be a minor but crucially necessary restriction (see also Section 5).

As mentioned, the conjectured optimal communication rate of $1 - O(\sqrt{\epsilon})$ for the oblivious adversarial setting is worse than the $1 - O(H(\epsilon))$ communication rate achievable in the one-way communication settings. However, the conjectured upper bound seems to be tight mainly for "maximally interactive" protocols, i.e., protocols in which the party that is sending bits changes frequently. In particular, *alternating* protocols, in which Alice and Bob take turns sending a single bit, seem to require the most redundancy for a noise-resilient encoding. On the other hand, the usual one-way communication case in which one party just sends a single message consisting of several bits is an example of a "minimally interactive" protocol. It is a natural question to consider what the tradeoff is between achievable communication rate and the level of interaction that takes place. In particular, most natural real-world protocols are rarely "maximally interactive" and could potentially be simulated with communication rates going well beyond $1 - O(\sqrt{\epsilon})$. We seek to investigate this possibility.

Our first contribution is to introduce the notion of *average message length* as a natural measure of the interactivity of a protocol in the context of analyzing communication rates. Loosely speaking, the average message length of an n-round protocol corresponds to the average number of bits a party sends before receiving a reply from the other party. A lower average message length roughly corresponds to more interactivity in a protocol, e.g., a maximally interactive protocol has average message length 1, while a one-way protocol with no interactivity has average message length n. The formal definition of average message length appears as Definition 3.1 in Section 3.

Our second and main contribution in this paper is to show that for protocols with an average message length of at least some constant in ϵ (but independent of the number of rounds n) one can go well beyond the $1 - \Theta(\sqrt{\epsilon})$ communication rate achieved by [11] for channels with oblivious adversarial errors. In fact, we show that for such protocols one can actually achieve a communication rate of $1 - \Theta(H(\epsilon))$, matching the communication rate for one-way communication up to the (unknown) constant in the $H(\epsilon)$ term.

Theorem 1.1. *For any $\epsilon > 0$ and any n-round interactive protocol Π with average message length $\ell = \Omega(\mathrm{poly}(1/\epsilon))$, it is possible to encode Π into a protocol over the same alphabet which, with probability at least $1 - \exp(-n\epsilon^6)$, simulates Π over an oblivious adversarial channel with an ϵ fraction of errors while achieving a communication rate of $1 - \Theta(H(\epsilon)) = 1 - \Theta(\epsilon \log(1/\epsilon))$.*

Under the (simplifying) assumption of *public shared randomness*, our protocol can furthermore be seen to have the nice property of being *rateless*. This means that the communication rate adapts automatically and only depends on the actual error rate ϵ without having to specify or know in advance what amount of noise to prepare for.

Theorem 1.2. *Suppose Alice and Bob have access to public shared randomness. For any $\epsilon' > 0$ and any n-round interactive protocol Π with average message length $\ell = \Omega(\mathrm{poly}(1/\epsilon'))$, it is possible to encode Π into protocol Π_{rateless} over the same alphabet such that for any* true error rate ϵ*, executing Π_{rateless} for $n(1 + O(H(\epsilon)) + O(\epsilon' \,\mathrm{polylog}(1/\epsilon')))$ rounds simulates Π with probability at least $1 - \exp(-n\epsilon'^3)$.*

We note that one should think of ϵ' in Theorem 1.2 as chosen to be very small, in particular, smaller than the smallest amount of noise one expects to encounter. In this case, the communication rate of the protocol simplifies to the optimal $1 - O(H(\epsilon))$ for essentially any $\epsilon > \epsilon'$. The only reason for not choosing ϵ' too small is that it very slightly increases the failure probability. As an example, choosing $\epsilon' = o(1)$ suffices to get ratelessness for any constant ϵ and still leads to an essentially exponential failure probability. Alternatively, one can even set $\epsilon' = n^{-1/6}$ which leads to optimal communication rates even for tiny sub-constant true error fractions $\epsilon > n^{-0.2}$ while still achieving a strong sub-exponential failure probability of at most $\exp(-\sqrt{n})$.

In Section 2, we introduce preliminaries about interactive coding. In Section 3, we describe the notion of average message length and its use in blocking. In Section 4, we provide some simple coding schemes for random channels as a warmup for our main result for the much more difficult case of adversarial channels. This leads up to Section 5, in which we describe some conceptual challenges and ideas needed to go beyond communication rates established in previous works. Finally, Section 6 provides a high-level description along with pseudocode of our interactive coding scheme, which allows us to obtain Theorems 1.1 and 1.2. Note that the proof of correctness of our coding scheme is omitted in this paper. A full analysis of our interactive coding scheme, which proves Theorems 1.1 and 1.2, is provided in the full version of this paper.

1.5 Further Related Works

Schulman was the first to consider the question of coding for interactive communication and showed that one can tolerate an adversarial error fraction of $\epsilon = 1/240$ with an unspecified constant communication rate [14, 15, 16]. Schulman's result also implies that for the easier setting of random errors, one can tolerate any error rate bounded away from 1/2 by repeating symbols multiple times. Since Schulman's seminal work, there has been a number of subsequent works pinning down the tolerable error fraction. For instance, Braverman and Rao [5] showed that any error fraction $\epsilon < 1/4$ can be tolerated in the realm of adversarial errors, provided that one can use larger alphabet sizes, and this bound was shown to be optimal. A series of subsequent works [4, 9, 10, 6, 7] worked to determine the error rate region under which non-zero communication rates can be obtained for a variety of models, e.g., adversarial errors, random errors, list-decoding, adaptivity, and channels with feedback. Unlike the initial coding schemes of [16] and [5] that relied on tree codes and as a result required exponential time computations, many of the newer coding schemes are computationally efficient [1, 3, 2, 8, 9]. All these results achieve small often unspecified constant communication rate of $\Theta(1)$ which is fixed and independent of amount of noise. Only the works of [12] and [11], which are already discussed above in Section 1.3 achieve a communication rate approaching 1 for error fractions going to zero.

2 Preliminaries

An *interactive protocol* Π consists of communication performed by two parties, Alice and Bob, over a channel with alphabet Σ. Alice has an input x and Bob has an input y, and the protocol consists of n *rounds*. During each round of a protocol, each party decides whether to listen or transmit a symbol from Σ, based on his input and the player's *transcript* thus far. Alice's *transcript* is defined as a tuple of symbols from Σ, one for each round that has occurred, such that the i^{th} symbol is either (a.) the symbol that Alice sent during the i^{th} round, if she chose to transmit, or (b.) the symbol that Alice received, otherwise.

Moreover, protocols can utilize *randomness*. In the case of *private randomness*, each party is given its own infinite string of independent uniformly random bits as part of its input. In the case of *shared randomness*, both parties have access to a common infinite random string during each round. In general, our protocols will utilize private randomness, unless otherwise specified.

In a *noiseless* setting, we can assume that in any round, exactly one party speaks and one party listens. In this case, the listening party simply receives the symbol sent by the speaking party.

The *communication order* of a protocol refers to the order in which Alice and Bob choose to speak or listen. A protocol is *non-adaptive* if the communication order is fixed prior to the start of the protocol, in which case, whether a party transmits or listens depends only on the round number. A simple type of non-adaptive protocol is an *alternating* protocol, in which one party transmits during odd numbered rounds, while the other party transmits during even numbered rounds. On the other hand, an *adaptive* protocol is one in which the communication order is not fixed prior to the start; therefore, the communication order can vary depending on the transcript of the protocol. In particular, each party's decision whether to speak or listen during a round will depend on his input, randomness, as well as the transcript of the protocol thus far.

For an n-round protocol over alphabet Σ, one can define an associated *protocol tree* of depth n. The protocol tree is a rooted tree in which each non-leaf node of the tree has $|\Sigma|$ children, and the outgoing edges are labeled by the elements of Σ. Each non-leaf node is owned by some player, and the owner of the node has a *preferred* edge that emanates from the node. The preferred edge is a function of the owner's input and any randomness that is allowed. Also, leaf nodes of the protocol tree correspond to ending states.

A proper execution of the protocol corresponds to the unique path from the root of the protocol tree to a leaf node, such that each traversed edge is the preferred edge of the parent node of the edge. In this case, each edge along the path can be viewed as a successive round in which the owner of the parent node transmits the symbol along the edge.

An example of a protocol tree is shown in Figure 1.

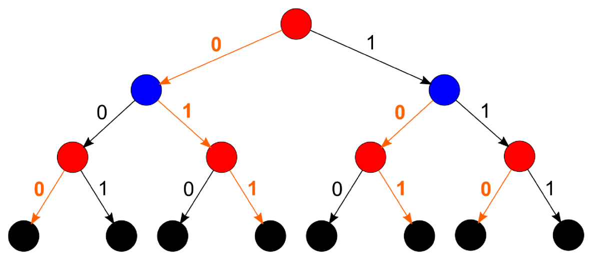

Figure 1: An example of a protocol tree for a 3-round interactive protocol. Nodes owned by Alice are colored red, while those owned by Bob are colored blue. Note that Alice always speaks during the first and third rounds, while Bob speaks during the second round. The orange edges are the set of preferred edges for some choice of inputs of Alice and Bob. In this case, a proper execution of the protocol corresponds to the path "011."

2.1 Communication Channels For our purposes, the communication between the two parties occurs over a *communication channel* that delivers a possibly corrupted version of the symbol transmitted by the sending party. In this work, transmissions will be from a *binary* alphabet, i.e., $\Sigma = \{0, 1\}$.

In a *random error channel*, each transmission occurs over a binary symmetric channel with crossover probability ϵ. In other words, in each round, if only one party is speaking, then the transmitted bit gets corrupted with probability ϵ.

This work mainly considers the *oblivious adversarial channel*, in which an adversary gets to corrupt at most ϵ fraction of the total number of rounds. However, the adversary is restricted to making his decisions prior to the start of the protocol, i.e., the adversary must decide which rounds to corrupt independently of the communication history and randomness used by Alice and Bob. For each round that the adversary decides to corrupt, he can either commit a *flip* error or *replace* error. Suppose a round has one party that speaks and one party that listens. Then, a flip error means that the listening party receives the opposite of the bit that the transmitting party sends. On the other hand, a replace error requires the adversary to specify a symbol $\alpha \in \Sigma$ for the round. In this case, the listening party receives α regardless of the symbol sent by the transmitting party.

An adaptive adversarial channel allows an adversary to corrupt at most ϵ fraction of the total number of rounds. However, in this case, the adversary does not have to commit to which rounds to corrupt prior to the start of the protocol. Rather, the adversary can decide to corrupt a round based on the communication history thus far, including what is being sent in the current round. Thus, in any round that the adversary chooses to corrupt in which one party transmits and one party receives, the adversary can make the listening party receive any symbol of his choice.

Note that we have not yet specified the behavior for rounds in which both parties speak or listen. Such rounds can occur for *adaptive* protocols when the communication occurs over a noisy communication channel.

If both parties speak during a round, we stipulate that neither party receives any symbol during that round (as neither party expects to receive a symbol).

Moreover, we stipulate that in rounds during which both parties listen, the symbols received by Alice and Bob are unspecified. In other words, an arbitrary symbol may be delivered to each of the parties, and we require that the protocol work for any choice of received symbols. Alternatively, one can imagine that the adversary chooses arbitrary symbols for Alice and Bob to receive without this being counted as a corruption (i.e., a free corruption that is not counted toward the budget of ϵ fraction of corruptions). The reason for this model is to disallow the possibility of transmitting information by using silence. An extensive discussion on the suitability of this error model can be found in [10].

3 Average Message Length and Blocked Protocols

One conceptual contribution of this work is to introduce the notion of *average message length* as a natural measure of the level of interactivity of a protocol. While this paper uses it only in the context of analyzing the optimal rate of interactive coding schemes, we believe that this notion and parametrization will also be useful in other settings, such as compression. Next, we define this notion formally.

Definition 3.1. *The* average message length ℓ *of an n-round interactive protocol* Π *is the minimum, over all paths in the protocol tree of* Π*, of the average length in bits of a maximal contiguous block (spoken by a single party) down the path.*

More precisely, given any string $s \in \{0,1\}^n$*, there exist integer message lengths* $l_0, \dots, l_k > 0$ *such that along the path of* Π *given by s one player (either Alice or Bob) speaks between round* $1 + \sum_{j<i} l_j$ *and round* $\sum_{j\leq i} l_j$ *for even i while the other speaks during the remaining intervals, i.e., those for odd i. We then define* ℓ_s *to be the average of these message lengths* $l_0, \dots, l_k$ *and define the* average message length *of* Π *to be minimum over all possible inputs, i.e.,* $\ell = \min_{s\in\{0,1\}^n} \ell_s$.

An alternate characterization of the amount of interaction in a protocol involves the number of alternations in the protocol:

Definition 3.2. *An n-round protocol* Π *is said to be* k-alternating *if any path in the protocol tree of* Π *can be divided into at most k blocks of consecutive rounds such that only one person (either Alice or Bob) speaks during each block.*

More precisely, Π *is* k-alternating *if, given any string* $s \in \{0,1\}^n$*, there exist* $k' \leq k$ *integers* $r_0, r_1, \dots, r_{k'}$ *with* $0 = r_0 < \dots < r_{k'} = n$*, such that along the path of* Π *given by s, only one player (either Alice or Bob) speaks for rounds* $r_i + 1, \dots, r_{i+1}$ *for any* $0 \leq i < k'$.

It is easy to see that the two notions are essentially equivalent, as an n-round protocol with average message length ℓ is an (n/ℓ)-alternating protocol, and a k-alternating n-round protocol has average message length n/k. Note that an n-round *alternating* protocol has average message length 1, while a *one-way* protocol has average message length n. The average message length can thus be seen as a natural measure for the interactivity of a protocol.

We emphasize that the average message length definition does not require message lengths to be uniform along any path or across paths. In particular, this allows for the length of a response to vary depending on what was communicated before, e.g., the statement the other party has just made—a common phenomenon in many applications. Taking as an example real-world conversations between two people, responses to statements can be as short as a simple "I agree" or much longer, depending on what the conversation has already covered and what the opinion or input of the receiving party is. Thus, a sufficiently large average message length roughly states that while the i^{th} response of a person can be short or long depending on the history of the conversation, no sequence of responses can lead to two parties going back and forth with super short statements for too long a period of time. This flexibility makes the average message length a highly applicable parameter that is reasonably large in most settings of interest. We expect it to be a very useful parametrization for questions going beyond the communication rate considered here.

However, the non-uniformity of protocols with an average message length bound can make the design and analysis of protocols somewhat harder than one would like. Fortunately, adding some dummy rounds of communication in a simple procedure we call *blocking* allows us to transform any protocol with small number of alternations into a much more regularly structured protocol which we refer to as *blocked*.

Definition 3.3. *An n-round protocol* Π *is said to be* b-blocked *if for any* $1 \leq j \leq \lceil n/b \rceil$*, only one person (either Alice or Bob) speaks during all rounds r such that* $(j-1)b < r \leq jb$.

Lemma 3.1. *Any n-round k-alternating protocol* Π *can be simulated by a b-blocked protocol* Π' *that consists of at most* $n + kb$ *rounds.*

We omit the proof of Lemma 3.1 here, but it can be found in the full version of this paper.

4 Warmup: Interactive Coding for Random Errors

As a warmup for the much more difficult adversarial setting, we first consider the setting of random errors, as this will illustrate several ideas including blocking, the use of error-correcting codes, and how to incorporate those with known techniques in coding for interactive communication.

In this section, we suppose that each transmission of Alice and Bob occurs over a binary symmetric channel with an ϵ probability of corruption. Recall that we wish to encode an n-round protocol Π into a protocol $\Pi_{\text{enc}}^{\text{random}}$ such that with high probability over the communication channel, execution of $\Pi_{\text{enc}}^{\text{random}}$ robustly simulates Π. By [11], it is known that one can achieve a communication rate of $1 - O(\sqrt{\epsilon})$. In this section, we show how to go beyond the rate of $1 - O(\sqrt{\epsilon})$ for protocols with at least a constant (in ϵ) average message length.

4.1 Trivial Scheme for Non-Adaptive Protocols with Minimum Message Length The first coding scheme we present for completeness is a completely trivial and straightforward application of error correcting codes which works for *non-adaptive* protocols Π with a guaranteed *minimum* message length. In particular, the coding scheme achieves a communication rate of $1 - O(H(\epsilon))$ for non-adaptive protocols with minimum message length $\Omega((1/\epsilon)\log n)$.

In particular, we assume that Π is a a non-adaptive n-round protocol with message lengths of size $b_1, b_2, \ldots, b_k$, i.e., Alice sends b_1 bits, then Bob sends b_2 bits, and so on. Moreover, we assume that that $b_1, b_2, \ldots, b_k \geq b$, where $b = \Omega((1/\epsilon)\log n)$ is the minimum message length.

Now, we can form the encoded protocol $\Pi_{\text{enc}}^{\text{random}}$ by simply having the transmitting party replace its intended message in Π (of b_i bits) with the encoding (of length, say, b'_i) of the message under an error-correcting code of minimum relative distance $\Omega(\epsilon)$ and rate $1 - O(H(\epsilon))$ and then transmitting the resulting codeword. The receiver then decodes the word according to the nearest codeword of the appropriate code.

Note that for any given message (codeword) of length b'_i, the expected number of corruptions due to the channel is $\epsilon b'_i$. Thus, by Chernoff bound, the probability that the corresponding codeword is corrupted beyond half the minimum distance of the relevant error-correcting code is $e^{-\Omega(\epsilon b')} = n^{-\Omega(1)}$. Since $k = O(n/b) = O(n\epsilon/\log n)$, the union bound implies that the probability that any of the $k < n$ messages is corrupted beyond half the minimum distance is also $n^{-\Omega(1)}$. Thus, with probability $1 - n^{-\Omega(1)}$, $\Pi_{\text{enc}}^{\text{random}}$ simulates the original protocol without error. Moreover, the overall communication rate is clearly $1 - O(H(\epsilon))$ due to the choice of the error-correcting codes.

Remark 4.1. *Note that the aforementioned trivial coding scheme has the disadvantage of working only for nonadaptive protocols with a certain* minimum *message length, which is a much stronger assumption than average message length. In Section 4.2, we show how to get around this problem by converting the input protocol to a* blocked *protocol.*

Another problem with the coding scheme is that the minimum message length is required to be $\Omega_\epsilon(\log n)$. This is in order to ensure that the probability of error survives a union bound, as the trivial coding scheme has no mechanism for recovering if a particular message gets corrupted. This also results in a success probability of only $1 - 1/\text{poly}(n)$ instead of the $1 - \exp(n)$ one would like to have for a coding scheme. Section 4.2 shows how to rectify both problems by combining the reduced error probability of a error correcting code failing with any existing interactive coding scheme, such as [11].

4.2 Coding Scheme for Protocols with Average Message Length of $\Omega(\log(1/\epsilon)/\epsilon^2)$ In this section, we build on the trivial scheme discussed earlier to provide an improved coding scheme that handles any protocol Π with an *average message length* of at least $\ell = \Omega(\log(1/\epsilon)/\epsilon^2)$.

The first step is to transform Π into a protocol that is blocked. Note that the Π is a k-alternating protocol, where $k = n/\ell = O(n\epsilon^2/\log(1/\epsilon))$. Thus, by Lemma 3.1, we can transform Π into a b-blocked protocol Π_{blk}, for $b = \Theta(\log(1/\epsilon)/\epsilon)$, such that Π_{blk} simulates Π and has $n_b = n + kb = n(1 + O(\epsilon))$ rounds.

Now, we view Π_{blk} as a q-ary protocol with n_b/b rounds, where $q = 2^b$. This can be done by grouping the symbols in each b-sized block as a single symbol from an alphabet of size q. Next, we can use the coding scheme of [11] in a blackbox manner to encode this q-ary protocol as a q-ary protocol Π' with $\frac{n_b}{b}(1 + \Theta(\sqrt{\epsilon'}))$ rounds such that Π' simulates Π under oblivious random errors with error fraction ϵ' (i.e., each q-ary symbol is corrupted (in any way) with an independent probability of at most ϵ'). We pick $\epsilon' = \epsilon^4$.

Finally, we transform Π' into a *binary* protocol $\Pi_{\text{enc}}^{\text{random}}$ as follows: We expand each q-ary symbol of Π' back into a sequence of b bits and then expand the b bits into $b' > b$ bits using an error-correcting code. In particular, we use an error-correcting code $\mathcal{C} : \{0,1\}^b \to \{0,1\}^{b'}$ with block length $b' = b + (2c + \delta)\log^2(1/\epsilon)$ and minimum distance $2c\log(1/\epsilon)$ for appropriate constants c, δ (such a code is guaranteed to exist by the Gilbert-Varshamov bound). Thus, $\Pi_{\text{enc}}^{\text{random}}$ is a b'-blocked binary protocol with $n_b \cdot \frac{b'}{b}(1 + \Theta(\sqrt{\epsilon'})) = n(1 + O(\epsilon\log(1/\epsilon))$ rounds. Moreover, each b'-sized block of $\Pi_{\text{enc}}^{\text{random}}$ simply simulates each q-ary symbol of Π' and the listening party simply decodes the received

b' bits to the nearest codeword of $\mathcal{C}$.

To see that $\Pi_{\text{enc}}^{\text{random}}$ successfully simulates Π in the presence of random errors with error fraction ϵ, observe that a b'-block is decoded incorrectly if and only if more than $d/2$ of the b' bits are corrupted. By the Chernoff bound, the probability of such an event is $< \epsilon^4$ (for appropriate choice of c, δ). Thus, since Π' is known to simulate Π under oblivious errors with error fraction ϵ^4, it follows that $\Pi_{\text{enc}}^{\text{random}}$ satisfies the desired property.

5 Conceptual Challenges and Key Ideas

In this section, we wish to provide some intuition for the difficulties in surpassing the $1 - \Theta(\sqrt{\epsilon})$ communication rate for interactive coding when dealing with non-random errors. We do this because the adversarial setting comes with a completely new set of challenges that are somewhat subtle but nonetheless fundamental. As such, the techniques used in the previous section for interactive coding under random errors still provide a good introduction to some of the building blocks in the framework we use to deal with the adversarial setting, but they are not sufficient to circumvent the main technical challenges. Indeed, we show in this section that the adversarial setting inherently requires several completely new techniques to beat the $1-\Theta(\sqrt{\epsilon})$ communication rate barrier.

We begin by noting that all existing interactive coding schemes encode the input protocol Π into a protocol Π' with a certain type of structure: There are some, *a priori* specified, communication rounds which simulate rounds of the original protocol (i.e., result in a walk down the protocol tree of Π), while other rounds constitute *redundant information* which is used for error correction. In the case of protocols that use hashing (e.g., [11], [12]), this is directly apparent in their description, as rounds in which hashes and control information are communicated constitute redundant information. However, this is also the case for all protocols based on tree codes (e.g., [5, 10, 9]): To see this, note that in such protocols, one can simply use an underlying tree code that is linear and systematic, with the non-systematic portion of the tree code then corresponding to redundant rounds.

We next present an argument which shows that, due to the above structure, no existing coding scheme can break the natural $1 - \Omega(\sqrt{\epsilon})$ communication rate barrier, even for protocols with near-linear $o(n)$ average message lengths. This will also provide some intuition about what is required to surpass this barrier.

Suppose that for a (randomized) n-round communication protocol Π, the simulating protocol Π' has the above structure and a communication rate of $1-\epsilon'$. The simulation Π' thus consists of exactly $N = n/(1 - \epsilon')$ rounds. Note that, since every simulation must have at least n non-redundant rounds, the fraction of redundant rounds in Π' can be at most ϵ'. Given that the position of the redundant rounds is fixed, it is therefore possible to find a window of $(\epsilon/\epsilon')N$ consecutive rounds in Π' which contain at most ϵN redundant rounds, i.e., an ϵ' fraction. Now, consider an oblivious adversarial channel that corrupts all the redundant information in the window along with a few extra rounds. Such an adversary renders any error correction technique useless, while the few extra errors derail the unprotected parts of the communication, thereby rendering essentially all the non-redundant information communicated in this window useless as well—all while corrupting essentially only ϵN rounds in total. This implies that in the remaining $N - (\epsilon/\epsilon')N$ communication rounds outside of this window, there must be at least n non-redundant rounds in order for Π' to be able to successfully simulate Π. However, it follows that $N - (\epsilon/\epsilon')N \geq n = N(1 - \epsilon')$ which simplifies to $1 - (\epsilon/\epsilon') \geq 1 - \epsilon'$, or $\epsilon'^2 \geq \epsilon$, implying that the communication rate of $1 - \epsilon'$ can be at most $1 - \Omega(\sqrt{\epsilon})$, where ϵ is the fraction of errors applied by the channel.

One can note that a main reason for the $1 - \Omega(\sqrt{\epsilon})$ limitation in the above argument is that the adversary can target the rounds with redundant information in the relevant window. For instance, in the interactive coding scheme of [11], the rounds with control information are in predetermined positions of the encoded protocol, and so, the adversary knows which locations to corrupt.

Our idea for overcoming the aforementioned limitations in the case of an *oblivious* adversarial channel is to use some type of **information hiding** to hide the locations of the redundant rounds carrying control/verification information. In particular, we randomize the locations of control information bits within the output protocol, which allows us to guard against attacks that target solely the redundant information. In order to allow for this synchronized randomization in the standard *private randomness* model assumed in this paper, Alice and Bob use the standard trick of running an error-corrected randomness exchange procedure that allows them to establish some shared randomness hidden from the oblivious adversary that can be used for the rest of the simulation. Note that this inherently does not work for a *fully adaptive* adversary, as the adversary can adaptively choose which locations to corrupt based on any randomness that has been shared over the channel. In fact, we believe that beating the $1 - \Omega(\sqrt{\epsilon})$ rate barrier against fully adaptive adversaries may be fundamentally impossible for precisely this reason.

Information hiding, while absolutely crucial, does not, however, make use of a larger average message length which, according to the conjectures of [11], is necessary to beat the $1 - \Omega(\sqrt{\epsilon})$ barrier. The idea we use for this, as already demonstrated in Section 4, is the use of blocking and the subsequent application of error-correcting codes on each such block.

Unfortunately, the same argument as given above shows that a straightforward application of *block* error-correcting codes, as done in Section 4, cannot work against an oblivious adversarial channel. The reason is that in such a case, an application of *systematic* block error-correcting codes would be possible as well, and such codes again have pre-specified positions of redundancy which can be targeted by the adversarial channel. In particular, one could again disable all redundant rounds including the non-systematic parts of block error-correcting codes in a large window of $(\epsilon/\epsilon')N$ rounds and make the remaining communication useless with few extra errors. More concretely, suppose that one simply encodes all blocks of data with a standard block error-correcting code. For such block codes, one needs to specify *a priori* how much redundancy should be added, and the natural direction would be to set the relative distance to, say, 100ϵ given that one wants to prepare against an error rate of ϵ. However, this would allow the adversary to corrupt a constant fraction (e.g., 1/200) of error correcting codes beyond their distance, thus making a constant fraction of the communicated information essentially useless. This would lead to a communication rate of $1 - \Theta(1)$. It can again be easily seen that in this tradeoff, the best fixed relative distance one can choose for block error-correcting codes is essentially $\sqrt{\epsilon}$, which would lead to a rate loss of $H(\sqrt{\epsilon})$ for the error-correcting codes but would also allow the adversary to corrupt at most a $\sqrt{\epsilon}$ fraction of all codewords. This would again lead to an overall communication rate of $1 - \tilde{\Omega}(\sqrt{\epsilon})$.

Our solution to the hurdle of having to commit to a fixed amount of redundancy in advance is to use **rateless error-correcting codes**. Unlike block error-correcting codes with fixed block length and minimum distance, rateless codes encode a message into a potentially *infinite* stream of symbols such that having access to enough uncorrupted symbols allows a party to decode the desired message with a resulting communication rate that *adapts* to the true error rate without requiring *a priori* knowledge of the error rate. Since it is not possible for Alice and Bob to know in advance which data bits the adversary will corrupt, rateless codes allow them to adaptively adjust the amount of redundancy for each communicated block, thereby allowing the correction of errors without incurring too great a loss in the overall communication rate.

6 Main Result: Interactive Coding for Oblivious Adversarial Errors

In this section, we develop our main result. We remind the reader that in the oblivious adversarial setting assumed throughout the rest of this paper, the adversary is allowed to corrupt up to an ϵ fraction of the total number of bits exchanged by Alice and Bob. The adversary commits to the locations of these bits before the start of the protocol. Alice and Bob will use randomness in their encoding, and one asks for a coding scheme that allows Alice and Bob to recover the transcript of the original protocol with exponentially high probability in the length of the protocol (over the randomness that Alice and Bob use) for any fixed error pattern chosen by the adversary.

For simplicity in exposition, we assume that the input protocol is *binary*, so that the simulating output protocol will also be binary. However, the results hold virtually as-is for protocols over larger alphabet. We first provide a high-level overview of our construction of an encoded protocol. The pseudocode of the algorithm appears in Figure 3.

6.1 High-Level Description of Coding Scheme

Let us describe the basic structure of our interactive coding scheme. Suppose Π is an n-round binary input protocol with average message length $\ell \geq \text{poly}(1/\epsilon)$. Using Lemma 3.1, we first produce a B-blocked binary protocol Π_{blk} with n' rounds that simulates Π.

Our encoded protocol $\Pi_{\text{enc}}^{\text{oblivious}}$ will begin by having Alice and Bob performing a *randomness exchange procedure*. More specifically, Alice will generate some number of bits from her private randomness and encode the random string using an error-correcting code of an appropriate rate and distance. Alice will then transmit the encoding to Bob, who can decode the received string. This allows Alice and Bob to maintain *shared random bits*. The randomness exchange procedure is described in further detail in Section 6.3.

Next, $\Pi_{\text{enc}}^{\text{oblivious}}$ will simulate the B-sized blocks (which we call *B-blocks*) of Π_{blk} in order in a structured manner. Each B-block will be encoded as a string of $2B$ bits using a *rateless code*, and the encoded string will be divided into *chunks* of size $b < B$. For a detailed discussion on the encoding procedure via rateless codes, see Section 6.4.

Now, $\Pi_{\text{enc}}^{\text{oblivious}}$ will consist of a series of N_{iter} *iterations*. Each iteration consists of transmitting b' rounds, and we call such a b'-sized unit a *mini-block*, where $b' > b$. Each mini-block will consist of b *data bits*, as well as $b' - b$ bits of *control information*. The data bits in successive mini-blocks will taken from the successive b-sized chunks obtained by the encoding under the rateless code. Meanwhile, the control information bits are sent by Alice and Bob in order to check whether they are in sync with each other and to allow a *backtracking* mechanism to tack place if they are not.

For a particular B-block that is being simulated, mini-blocks keep getting sent until the receiving party of the B-block is able to decode the correct B-block, after which Alice and Bob move on to the next B-block in Π.

In addition to data bits, each mini-block also contains $b' - b$ bits of control information. A party's unencoded control information during a mini-block consists of some hashes of his view of the current state of the protocol as well as some backtracking parameters. The aforementioned quantities are encoded using a hash for verification as well as an error-correcting code. Each party sends his encoded control information as part of each mini-block. The locations of the control information within each mini-block will be randomized for the sake of *information hiding*, using bits from the shared randomness of Alice and Bob. This is described in further detail in Section 6.5. Moreover, we note that the hashes used for the control information in each mini-block are seeded using bits from the shared randomness. The structure of each mini-block is shown in Figure 2.

After each iteration, Alice and Bob try to decode each other's control information in order to determine whether they are in sync. If not, the parties decide whether to backtrack in a controlled manner (see Section 6.6 for details).

Throughout the protocol, Alice maintains a *block index* c_A (which indicates which block of Π_{blk} she believes is currently being simulated), a *chunk counter* j_A, a *transcript* (of the blocks in Π_{blk} that have been simulated so far) T_A, a *global counter* m (indicating the number of the current iteration), a *backtracking parameter* k_A, as well as a *sync parameter* sync_A. Similarly, Bob maintains c_B, j_B, T_B, m, k_B, and sync_B.

6.2 Parameters We now set the parameters of the protocol. For convenience, we will define a *loss parameter* $\epsilon' < \epsilon$. Our interactive coding scheme will incur a rate loss of $\Theta(\epsilon' \operatorname{polylog}(1/\epsilon'))$, in addition to the usual rate loss of $\Theta(H(\epsilon))$. Alice and Bob are free to decide on an ϵ' based on what rate loss they are willing to tolerate in the interactive coding scheme. In particular, note that if $\epsilon' = \Theta(\epsilon^2)$, then the rate loss of $\Theta(\epsilon' \operatorname{polylog}(1/\epsilon'))$ is overwhelmed by $\Theta(H(\epsilon))$. For the purposes of Theorem 1.1, it will suffice to take $\epsilon' = \Theta(\epsilon^2)$ at then end, but for the sake of generality, we maintain ϵ' as a separate parameter.

We now take the average message length threshold to be $\Omega(1/\epsilon'^3)$, i.e., we assume that our input protocol Π has average message length $\ell = \Omega(1/\epsilon'^3)$. Then, Π has at most $\mathsf{alt} = n/\ell = O(n\epsilon'^3)$ alternations. Moreover, we take $B = \Theta(1/\epsilon'^2)$ and $b = s = \Theta(1/\epsilon')$, with $B = sb$. Then, by Lemma 3.1, note that $n' \leq n + \mathsf{alt} \cdot B = n(1 + O(\epsilon'))$.

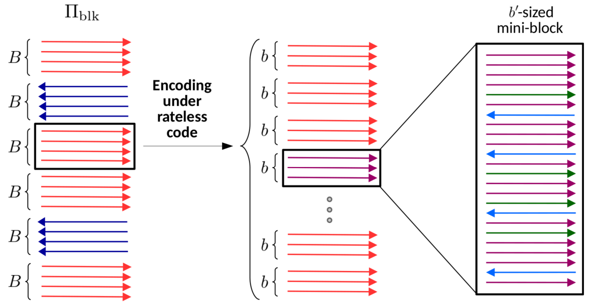

Figure 2: Each B-block of Π_{blk} gets encoded into chunks of size b using a rateless code. Every b'-sized mini-block in $\Pi_{\text{enc}}^{\text{oblivious}}$ consists of the b bits of such a chunk, along with $(b' - b)/2$ bits of Alice's control information and $(b' - b)/2$ bits of Bob's control information. The positions of the control information within a mini-block are randomized. Note that rounds with Alice's control information are in green, while rounds with Bob's control information are in light blue.

We also take $b' = b + 2c \log(1/\epsilon')$, so that within each b'-sized mini-block, each party transmits $c \log(1/\epsilon')$ bits of (encoded) control information.

Finally, we take $N_{\mathsf{iter}} = \frac{n'}{b}(1 + \Theta(\epsilon \log(1/\epsilon))$ iterations. This will guarantee, with high probability, that at the end of the protocol, Alice and Bob have successfully simulated all blocks of Π_{blk}, and therefore, Π. Also, it should be noted that we append trivial blocks of zeros (sent by, say, Alice) to the end of Π_{blk} to simulate in case $\Pi_{\text{enc}}^{\text{oblivious}}$ ever runs out of blocks of Π_{blk} to simulate (because it has reached the bottom of the protocol tree) before N_{iter} iterations of $\Pi_{\text{enc}}^{\text{oblivious}}$ have been executed.

6.3 Randomness Exchange Alice and Bob will need to have some number of shared random bits throughout the course of the protocol. The random bits will be used for two main purposes: *information hiding* and *seeding hash functions*, which will be discussed in Section 6.5. As it turns out, it will suffice for Alice and Bob to have $l' = O(n\epsilon' \operatorname{polylog}(1/\epsilon'))$ shared random bits for the entirety of the protocol, using some additional tricks.

Thus, in the private randomness model, it suffices for Alice to generate the necessary number of random bits and transmit them to Bob using an error-correcting code. More precisely, Alice generates a uniformly random string $\mathsf{str} \in \{0,1\}^{l'}$, uses an error-correcting code $\mathcal{C}^{\mathsf{exchange}} : \{0,1\}^{l'} \to \{0,1\}^{10\epsilon N_{\mathsf{iter}} b'}$ of relative distance $2/5$ to encode str, and transmits the encoded string to Bob. Since the adversary can corrupt only at most ϵ fraction of all bits, the transmitted string cannot be corrupted beyond half the minimum distance of $\mathcal{C}^{\mathsf{exchange}}$. Hence, Bob can decode the received string and determine str.

Note that the exchange of randomness via the codeword in $\mathcal{C}^{\mathsf{exchange}}$ results in a rate loss of $\Theta(\epsilon)$, which is still overwhelmed by $\Theta(H(\epsilon))$.

6.4 Sending Data Bits Using "Rateless" Error-Correcting Codes To transmit data from blocks of Π_{blk}, we will use an error-correcting code that has incremental distance properties. One can think of this as a rateless code with minimum distance properties. Recall that $b = s = \Theta(1/\epsilon')$ and $B = sb$. In particular, we require an error-correcting code $\mathcal{C}^{\mathsf{rateless}} : \{0,1\}^B \to \{0,1\}^{2B}$ for which the output is divided in to $2s$ chunks of b bits each such that the code restricted to any contiguous (cyclic) block of $> s$ chunks has a certain guaranteed minimum distance. The following lemma guarantees the existence of such a code. We omit the proof, but it appears in the full version of this paper.

LEMMA 6.1. *For sufficiently large b, s, there exists an error-correcting code $\mathcal{C} : \{0,1\}^{sb} \to \{0,1\}^{2sb}$ such that for any $a = 0, 1, \ldots, 2s-1$ and $j = s+1, s+2, \ldots, 2s$, the code $\mathcal{C}_{a,j} : \{0,1\}^{sb} \to \{0,1\}^{jb}$ formed by restricting $\mathcal{C}$ to the bits $ab, ab+1, \ldots, ab+jb-1$ (modulo $2sb$) has relative distance at least $\delta_j = H^{-1}\left(\frac{j-s}{j} - \frac{1}{4s}\right)$, while $\mathcal{C}$ has relative distance at least $\delta_{2s} = \frac{1}{15}$. (Here, H^{-1} is the unique inverse of H that takes values in $[0, 1/2]$.)*

REMARK 6.1. *We set $b = s = \Theta(1/\epsilon')$. Thus, for suitably small $\epsilon' > 0$, there exists such a code $\mathcal{C}$ as guaranteed by Lemma 6.1. Moreover, it is possible to find such a code by brute force in time $\text{poly}(1/\epsilon')$.*

Thus, Alice and Bob can agree on a fixed error-correcting code $\mathcal{C}^{\mathsf{rateless}}$ of the type guaranteed by Lemma 6.1 prior to the start of the algorithm. Now, let us describe how data bits are sent during the iterations of $\Pi_{\text{enc}}^{\text{oblivious}}$. The blocks of $\Pi_{\text{enc}}^{\text{oblivious}}$ are simulated in order as follows.

First, suppose Alice's block index c_A indicates a B-block in Π_{blk} during which Alice is the sender. Then in $\Pi_{\text{enc}}^{\text{oblivious}}$, Alice will transmit up to a maximum of $2s$ chunks (of size b) that will encode the data x from that block. More specifically, Alice will compute $y = \mathcal{C}^{\mathsf{rateless}}(x) \in \{0,1\}^{2B}$ and decompose it as $y = y_0 \circ y_1 \circ \cdots \circ y_{2s-1}$, where $\circ$ denotes concatenation and $y_0, y_1, \ldots, y_{2s-1} \in \{0,1\}^b$.

Recall that each mini-block of $\Pi_{\text{enc}}^{\text{oblivious}}$ contains b data bits (in addition to $b' - b$ control bits). Thus, Alice can send each y_i as the data bits of a mini-block. The chunk that Alice sends in a given iteration depends on the global counter m. In particular, Alice always sends the chunk $y_{m \bmod 2s}$. Moreover, Alice keeps a chunk counter j_A, which is set to 0 during the first iteration in which she transmits a chunk from y and then increases by 1 during each subsequent iteration (until $j_A = 2s$, at which point j_A stops increasing).

On the other hand, suppose Alice's block index c_A indicates a B-block in Π_{blk} during which Alice is the receiver. Then, Alice listens for data during each mini-block. Alice stores her received b-sized chunks as $\widetilde{g}_0, \widetilde{g}_1, \ldots$ and increments her chunk counter j_A after each iteration to keep track of how many chunks she has stored, along with a, an index indicating which y_a she expects the first chunk $\widetilde{g}_0$ to be. Once Alice has received more than s chunks (i.e., $j_A > s$), she starts to keep an estimate $\widetilde{x}$ of the data x that Bob is sending that Alice has by decoding $\widetilde{g}_0 \circ \widetilde{g}_1 \circ \cdots \circ \widetilde{g}_{j_A-1}$ to the nearest codeword of $\mathcal{C}_{a,j_A}^{\mathsf{rateless}}$. This estimate is updated after each subsequent iteration. As soon as Alice undergoes an iteration in which she receives valid control information suggesting that $\widetilde{x} = x$ (if Alice's estimate $\widetilde{x}$ matches the hash of x that Bob sends as control information, see Section 6.5), she advances her block index c_A and appends her transcript T_A with $\widetilde{x}$.

Note that it is possible that j_A reaches $2s$ and Alice has not yet received valid control information suggesting that he has decoded x. In this case, Alice resets j_A to 0 and also resets a to the current value of m, thereby restarting the listening process. Also, during any iteration, if Alice receives control information suggesting that $j_B < j_A$ (i.e., Alice has been listening for a greater number of iterations than Bob has been transmitting), then again, Alice resets j_A and a and restarts the process.

REMARK 6.2. *Using a rateless code allows the amount of redundancy in data that the sender sends to* adapt *to the number of errors introduced by the adversary, rather than wasting extra bits or not sending enough of them.*

6.5 Control Information Alice's unencoded control information in the mth iteration consists of (1.) a hash $h_{A,c}^{(m)} = hash(c_A, S)$ of the block index c_A, (2.) a hash $h_{A,x}^{(m)} = hash(x, S)$ of the data in the current block of Π_{blk} being communicated, (3.) a hash $h_{A,k}^{(m)} = hash(k_A, S)$ of the *backtracking parameter* k_A, (4.) a hash $h_{A,T}^{(m)} = hash(T_A, S)$ of Alice's transcript T_A, (5.) a hash $h_{A,\mathtt{MP1}}^{(m)} = hash(T_A[1, \mathtt{MP1}], S)$ of Alice's transcript up till the first *meeting point*, (6.) a hash $h^{(m)}_{A,\mathtt{MP2}} = hash(T_A[1, \mathtt{MP2}], S)$ of Alice's transcript up till the second *meeting point*, (7.) the chunk counter j_A, and (8.) the *sync parameter* sync_A. Here, S refers to a string of fresh random bits used to seed the hash functions (note that S is different for each instance). Thus, we write Alice's unencoded control information as

$$\begin{aligned}&\mathsf{ctrl}^{(m)}_A\\ &= \left(h^{(m)}_{A,c}, h^{(m)}_{A,x}, h^{(m)}_{A,k}, h^{(m)}_{A,T}, h^{(m)}_{A,\mathtt{MP1}}, h^{(m)}_{A,\mathtt{MP2}}, j_A, \mathsf{sync}_A\right).\end{aligned}$$

Bob's unencoded control information $\mathsf{ctrl}^{(m)}_B$ is similar in the analogous way.

For the individual hashes, we can use the following Inner Product hash function $hash : \{0,1\}^l \times \{0,1\}^r \to \{0,1\}^p$, where $r = lp$:

$$\begin{aligned}hash(X,R) = \big(&\langle X, R_{[1,l]}\rangle, \langle X, R_{[l+1,2l]}\rangle,\\ &\ldots, \langle X, R_{[lp-(l-1),lp]}\rangle\big),\end{aligned}$$

where X is the quantity to be hashed, and R is a random seed. This choice of hash function guarantees the following property:

PROPERTY 6.1. *For any $X, Y \in \{0,1\}^l$ such that $X \neq Y$, we have that $\Pr_{R\sim\mathrm{Unif}(\{0,1\}^r)}[hash(X,R) = hash(Y,R)] \leq 2^{-p}$.*

Now, we wish to take output size $p = O(\log(1/\epsilon'))$ for each of the hashes so that the total size of each party's control information in any iteration is $O(\log(1/\epsilon'))$. Note that some of the quantities we hash (e.g., T_A, T_B) actually have size $l = \Omega(n)$. Thus, for the corresponding hash function, we would naively require $r = lp = \Omega(n \log(1/\epsilon'))$ fresh bits of randomness for the seed (per iteration), for a total of $\Omega(N_{\mathsf{iter}} n \log(1/\epsilon'))$ bits of randomness. However, as described in Section 6.3, Alice and Bob only have access to $O(n\epsilon' \mathrm{polylog}(1/\epsilon'))$ bits of shared randomness!

To get around this problem, we make use of δ-biased sources to minimize the amount of randomness we need. In particular, we can use the δ-biased sample space of [13] to stretch $\Theta(\log(L/\delta))$ independent random bits into a string of $L = \Theta(N_{\mathsf{iter}} n \log(1/\epsilon'))$ pseudorandom bits that are δ-biased. We take $\delta = 2^{-\Theta(N_{\mathsf{iter}}\cdot p)}$. The sample space guarantees that the L pseudorandom bits are $\delta^{\Theta(1)}$-statistically close to being k-wise independent for $k = \log(1/\delta) = \Theta(N_{\mathsf{iter}} \cdot p) = \Theta(N_{\mathsf{iter}} \log(1/\epsilon'))$. Moreover, the Inner Product Hash Function satisfies the following modified collision property, which follows trivially from Property 6.1 and the definition of δ-bias:

PROPERTY 6.2. *For any $X, Y \in \{0,1\}^l$ such that $X \neq Y$, we have that $\Pr_R[hash(X,R) = hash(Y,R)] \leq 2^{-p} + \delta$, where R is sampled from a δ-biased source.*

As it turns out, this property is good enough for our purposes. Thus, after the randomness exchange, Alice and Bob can simply take $\Theta(\log(L/\delta))$ bits from str and stretch them into an L-bit string $\mathsf{str}_{\mathrm{stretch}}$ as described. Then, for each iteration, Alice and Bob can simply seed their hash functions using bits from $\mathsf{str}_{\mathrm{stretch}}$.

6.5.1 Encoding and Decoding Control Information

In this section, we describe the encoding and decoding functions that Alice and Bob use for their control information. We start by listing the desired properties:

DEFINITION 6.1. *Suppose $X \in \{0,1\}^l$ and $V \in \{*, \neg, 0, 1\}^l$ for some $l > 0$. Then, we define $\mathsf{Corrupt}_V(X) = Y \in \{0,1\}^l$ as follows:*

$$Y_i = \begin{cases} V_i & \text{if } V_i \in \{0,1\} \\ X_i \oplus 1 & \text{if } V_i = \neg \\ X_i & \text{if } V_i = * \end{cases}.$$

Moreover, we define $\mathsf{wt}(V)$ to be the number of coordinates of V that are not equal to $*$.

REMARK 6.3. *Note that V corresponds to an error pattern, in which $*$ indicates a position that is not corrupted, while $\neg$ indicates a bit flip, and 0/1 indicates a bit that is fixed to the appropriate symbol (see Section 2.1 for details about* flip *and* replace *errors). The function $\mathsf{Corrupt}_V$ applies the error pattern V to the bit string given as an argument. Also, $\mathsf{wt}(V)$ is equal to the number of positions that are targeted for corruption.*

We require a seeded encoding function $\mathsf{Enc} : \{0,1\}^l \times \{0,1\}^r \to \{0,1\}^o$ as well as a seeded decoding function $\mathsf{Dec} : \{0,1\}^o \times \{0,1\}^r \to \{0,1\}^l \cup \{\perp\}$ such that the following property holds:

PROPERTY 6.3. *The following holds:*

1. *For any $X \in \{0,1\}^l$, $R \in \{0,1\}^r$, and $V \in \{*, \neg, 0, 1\}^o$ such that $\mathsf{wt}(V) < \frac{1}{8}o$,*

$$\mathsf{Dec}(\mathsf{Corrupt}_V(\mathsf{Enc}(X,R)), R) = X.$$

2. *For any $X \in \{0,1\}^l$ and $V \in \{0,1\}^o$ such that $\mathsf{wt}(V) \geq \frac{1}{8}o$,*

$$\begin{aligned}\Pr_{R\sim\mathrm{Unif}(\{0,1\}^r)}&[\mathsf{Dec}(\mathsf{Corrupt}_V(\mathsf{Enc}(X,R)), R)\\ &\notin \{X, \perp\}] \leq 2^{-\Omega(l)}.\end{aligned}$$

REMARK 6.4. *The second argument of* Enc *and* Dec *will be a* seed, *generated by taking r fresh bits from the shared randomness of Alice and Bob. A decoding output of $\perp$ indicates a decoding failure. Moreover, (1.) of*

Property 6.3 guarantees that a party can successfully decode the other party's control information if at most a constant fraction of the encoded control information bits are corrupted. On the other hand, (2.) of Property 6.3 guarantees that if a larger fraction of the encoded control information bits are corrupted, then the decoding party can detect any possible corruption with high probability.

We now exhibit Enc, Dec that satisfy Property 6.3. Our Enc consists of a three-stage encoding: (1.) append a hash value to the unencoded control information, (2.) encode the resulting string using an error-correcting code, and (3.) XOR each output bit with a fresh random bit taken from the shared randomness.

For our purposes, we want $l = O(\log(1/\epsilon'))$ to be the number of bits in $\mathsf{ctrl}_A^{(m)}$ (or $\mathsf{ctrl}_B^{(m)}$) and $o = c\log(1/\epsilon')$. First, we choose a hash function $h : \{0,1\}^l \times \{0,1\}^t \to \{0,1\}^{o'}$ that has the following property:

Property 6.4. *Suppose $X, U \in \{0,1\}^l$, where U is not the all-zeros vector, and $W \in \{0,1\}^{o'}$. Then,*

$$\Pr_{R\sim\text{Unif}(\{0,1\}^t)}[h(X+U,R) = h(X,R) + W] \le 2^{-o'}.$$

In particular, we can use the simple Inner Product Hash Function with $t = l \cdot o'$ and $o' = \Theta(\log(1/\epsilon'))$:

$$h(X,R) = \left(\left\langle X, R_{[1,l]}\right\rangle, \left\langle X, R_{[l+1,2l]}\right\rangle, \ldots, \left\langle X, R_{[l\cdot o'-(l-1), l\cdot o']}\right\rangle\right).$$

Next, we choose a *linear* error-correcting code $\mathcal{C}^{\mathsf{hash}} : \{0,1\}^{l+o'} \to \{0,1\}^o$ of constant relative distance 1/4 and constant rate. We take $r = t + o$ and define Enc as

$$\mathsf{Enc}(X,R) = \mathcal{C}^{\mathsf{hash}}(X \circ h(X, R_{[o+1,r]})) \oplus R_{[1,o]}.$$

Moreover, we define Dec as follows: Given Y, R, let X' be the decoding of $Y + R_{[1,o]}$ under $\mathcal{C}^{\mathsf{hash}}$ (using the nearest codeword of $\mathcal{C}^{\mathsf{hash}}$ and then inverting the map $\mathcal{C}^{\mathsf{hash}}$). We then define

$$\mathsf{Dec}(Y,R) = \begin{cases} X'_{[1,l]} & \text{if } h(X'_{[1,l]}, R_{[o+1,r]}) = X'_{[l+1,l+o']} \\ \perp & \text{if } h(X'_{[1,l]}, R_{[o+1,r]}) \neq X'_{[l+1,l+o']} \end{cases}.$$

As it turns out, the above functions Enc and Dec satisfy Property 6.3. We do not prove this fact here, but a proof appears in the full version of this paper.

6.5.2 Information Hiding

We now describe how the encoded control information bits are sent within each mini-block. Recall that in the m^{th} iteration, Alice chooses a fresh random seed R^A taken from the shared randomness str and computes her encoded control information $\mathsf{Enc}(\mathsf{ctrl}_A^{(m)}, R^A)$. Similarly, Bob chooses R^B and computes $\mathsf{Enc}(\mathsf{ctrl}_B^{(m)}, R^B)$. Recall that R^A, R^B are known to both Alice and Bob.

As discussed previously, the control information bits in each mini-block are not sent contiguously. Rather, the locations of the control information bits within each b'-sized mini-block are hidden from the oblivious adversary by using the shared randomness to agree on a designated set of $2c\log(1/\epsilon')$ locations. In particular, the locations of the control information bits sent by Alice and Bob during the m^{th} iteration are given by the variables $z_{m,i}^A$ and $z_{m,i}^B$ ($i = 1, \ldots, c\log(1/\epsilon')$), respectively. For each m, these variables are chosen randomly at the beginning using $O(\log^2(1/\epsilon'))$ fresh random bits from the preshared string str. Since there are N_{iter} iterations, this will require a total of $\Theta(N_{\mathsf{iter}} \cdot \log^2(1/\epsilon')) = \Theta(n\epsilon' \log^2(1/\epsilon'))$ random bits from str.

Thus, Alice sends the $c\log(1/\epsilon')$ bits of $\mathsf{Enc}(\mathsf{ctrl}_A^{(m)}, R^A)$ in positions $z_{m,i}^A$ ($i = 1, \ldots, c\log(1/\epsilon')$) of the mini-block of the m^{th} iteration, and similarly, Bob sends the bits of $\mathsf{Enc}(\mathsf{ctrl}_B^{(m)}, R^B)$ in positions $z_{m,i}^B$ ($i = 1, \ldots, c\log(1/\epsilon')$). Meanwhile, Bob listens for Alice's encoded control information in positions $z_{m,i}^A$ of the mini-block and assembles the received bits as a string $Y \in \{0,1\}^{c\log(1/\epsilon')}$, after which Bob tries to decode Alice's control information by computing $\mathsf{Dec}(Y, R^A)$. Similarly, Alice listens for Bob's encoded control information in locations $z_{m,i}^B$ and tries to decode the received bits.

After each iteration, Alice and Bob use their decodings of each other's control information to decide how to proceed. This is described in detail in Section 6.6.

6.6 Flow of the Protocol and Backtracking

Throughout $\Pi_{\text{enc}}^{\text{oblivious}}$, each party maintains a state that indicates whether both parties are in sync as well as parameters that allow for backtracking in the case that the parties are not in sync. After each iteration, Alice and Bob use their decodings of the other party's control information from that iteration to update their states. We describe the flow of the protocol in detail.

Alice and Bob maintain binary variables sync_A and sync_B, respectively, which indicate the players' individual perceptions of whether they are in sync. Note that $\mathsf{sync}_A = 1$ implies $k_A = 1$ (and similarly, $\mathsf{sync}_B = 1$ implies $k_B = 1$). Also, in the case that $\mathsf{sync}_A = 1$ (resp. $\mathsf{sync}_B = 1$), the variable speak_A (resp. speak_B) indicates whether Alice (resp. Bob) speaks in the c_A^{th} (resp. c_B^{th}) block of Π_{blk}, based on the transcript thus far.

Let us describe the protocol from Alice's point of view, as Bob's procedure is analogous. Note that after each iteration, Alice attempts to decode Bob's control information for that iteration. We say that

Alice *successfully decodes* Bob's control information if the decoding procedure (see Section 6.5.1) does not output $\perp$. In this case, we write the output of the control information decoder (for the m^{th} iteration) as

$$\widetilde{\mathsf{ctrl}}_B^{(m)} = \left(\widetilde{h}_{B,c}^{(m)}, \widetilde{h}_{B,x}^{(m)}, \widetilde{h}_{B,k}^{(m)}, \widetilde{h}_{B,T}^{(m)}, \widetilde{h}_{B,\mathtt{MP1}}^{(m)}, \widetilde{h}_{B,\mathtt{MP2}}^{(m)}, \widetilde{j}_B, \widetilde{\mathsf{sync}}_B\right).$$

We now split into two cases: $\mathsf{sync}_A = 1$ and $\mathsf{sync}_A = 0$.

Case 1 ($\mathsf{sync}_A = 1$): The general idea is that whenever Alice thinks she is in sync with Bob (i.e., $\mathsf{sync}_A = 1$), she either (a.) *listens* for data bits from Bob while updating her estimate $\widetilde{x}$ of block c_A of Π_{blk}, if $\mathsf{speak}_A = 0$, or (b.) *transmits*, as data bits of the next iteration, the (m mod $2s$)-th chunk of the encoding of x (the c_A-th B-block of Π_{blk}) under $\mathcal{C}^{\mathsf{rateless}}$, if $\mathsf{speak}_A = 1$ (see Section 6.4 for details).

If Alice is *listening* for data bits, then Alice expects that $k_A = k_B = 1$ and either (1.) $c_A = c_B$, $T_A = T_B$ or (2.) $c_A = c_B + 1$, $T_B = T_A[1 \ldots (c_B - 1)B]$. Condition (1.) is expected to hold if Alice has still not managed to decode the B-block x that Bob is trying to relay, while (2.) is expected if Alice has managed to decode x and has advanced her transcript but Bob has not yet realized this.

On the other hand, if Alice is *transmitting* data bits, then Alice expects that $k_A = k_B = 1$, as well as either (1.) $c_A = c_B, T_A = T_B$, or (2.) $c_B = c_A+1, T_B = T_A \circ x$, or (3.) $c_A = c_B+1, T_B = T_A[1 \ldots (c_B-1)B]$. Condition (1.) is expected to hold if Bob is still listening for data bits and has not yet decoded Alice's x, while (2.) is expected to hold if Bob has already managed to decode x and advanced his block index and transcript, and (3.) is expected to hold if Bob has been transmitting data bits to Alice (for the $(c_A - 1)$-th B-block of Π_{blk}), but Bob has not realized that Alice has decoded the correct B-block and moved on.

Now, if Alice manages to successfully decode Bob's control information in the most recent iteration, then Alice checks whether the hashes $\widetilde{h}_{B,c}^{(m)}$, $\widetilde{h}_{B,k}^{(m)}$, $\widetilde{h}_{B,T}^{(m)}$, $\widetilde{h}_{B,x}^{(m)}$, as well as $\widetilde{\mathsf{sync}}_B$ are consistent with Alice's expectations (as outlined in the previous two paragraphs). If not, then Alice sets $\mathsf{sync}_A = 0$. Otherwise, Alice proceeds normally.

REMARK 6.5. *Note that in general, if a party is trying to transmit the contents x of a B-block and the other party is trying to listen for x, then there is a delay of at least one iteration between the time that the listening party decodes x and the time that the transmitting party receives control information suggesting that the other party has decoded x. However, since $b/B = O(\epsilon')$, the rate loss due to this delay turns out to be just $O(\epsilon')$.*

Case 2 ($\mathsf{sync}_A = 0$): Now, we consider what happens when Alice believes she is out of sync (i.e., $\mathsf{sync}_A = 0$). In this case, Alice uses a meeting point based backtracking mechanism along the lines of [14] and [11]. We sketch the main ideas below.

Specifically, Alice keeps a backtracking parameter k_A that is initialized as 1 when Alice first believes she has gone out of sync and increases by 1 each iteration thereafter. (Note that k_A is also maintained when $\mathsf{sync}_A = 1$, but it is always set to 1 in this case.) Alice also maintains a counter E_A that counts the number of discrepancies between k_A and k_B, as well as *meeting point counters* v_1 and v_2. The counters E_A, v_1, v_2 are initialized to zero when Alice first sets sync_A to 0.

The parameter k_A measures the amount by which Alice is willing to backtrack in her transcript T_A. More specifically, Alice creates a *scale* $\widetilde{k}_A = 2^{\lfloor \log_2 k_A \rfloor}$ by rounding k_A to the largest power of two that does not exceed it. Then, Alice defines two *meeting points* `MP1` and `MP2` on this scale to be the two largest multiples of $\widetilde{k}_A B$ not exceeding $|T_A|$. More precisely, $\mathtt{MP1} = \widetilde{k}_A B \left\lfloor \frac{|T_A|}{\widetilde{k}_A B} \right\rfloor$ and $\mathtt{MP2} = \mathtt{MP1} - \widetilde{k}_A B$. Alice is willing to rewind her transcript to either one of $T_A[1 \ldots \mathtt{MP1}]$ and $T_A[1 \ldots \mathtt{MP2}]$, the last two positions in her transcript where the number of B-blocks of Π_{blk} that have been simulated is an integral multiple of $\widetilde{k}_A$.

If Alice is able to successfully decode Bob's control information, then she checks $\widetilde{h}_{B,k}^{(m)}$. If it does not agree with the hash of k_A, then Alice increments E_A. She also increments E_A if $\widetilde{\mathsf{sync}}_B = 1$.

Otherwise, if $\widetilde{h}_{B,k}^{(m)}$ matches her computed hash of k_A, then Alice checks whether either of $\widetilde{h}_{B,\mathtt{MP1}}^{(m)}, \widetilde{h}_{B,\mathtt{MP2}}^{(m)}$ matches the appropriate hash of $T_A[1 \ldots \mathtt{MP1}]$. If so, then Alice increments her counter v_1, which counts the number of times her *first* meeting point matches one of the meeting points of Bob. If not, then Alice then checks whether either of $\widetilde{h}_{B,\mathtt{MP1}}^{(m)}, \widetilde{h}_{B,\mathtt{MP2}}^{(m)}$ matches the hash of $T_A[1 \ldots \mathtt{MP2}]$ and if so, she increments her counter v_2, which counts the number of times her *second* meeting point matches one of the meeting points of Bob.

If Alice is not able to successfully decode Bob's control information from the most recent iteration (i.e., the decoder outputs $\perp$), she increments E_A.

Regardless of which of the above scenarios holds, Alice then increments k_A and updates $\widetilde{k}_A$, `MP1`, `MP2`.

Next, Alice checks whether to initiate a *transition.* Alice only considers making a transition if $k_A = \widetilde{k}_A \geq 2$ (i.e., k_A is a power of two and is ≥ 2). Alice first decides whether to initiate a *meeting point transition.* If $v_1 \geq 0.2k_A$, then Alice rewinds T_A to $T_A[1 \ldots \mathtt{MP1}]$ and resets $k_A, \widetilde{k}_A, \mathsf{sync}_A$ to 1 and E_A, v_1, v_2 to 0. Otherwise, if $v_2 \geq 0.2k_A$, then Alice rewinds T_A to $T_A[1 \ldots \mathtt{MP2}]$

and again resets $k_A, \widetilde{k}_A, \mathsf{sync}_A$ to 1 and E_A, v_1, v_2 to 0.

If Alice has not made a meeting point transition, then Alice checks whether $E_A \geq 0.2k_A$. If so, Alice undergoes an *error transition*, in which she simply resets $k_A, \widetilde{k}_A, \mathsf{sync}_A$ to 1 and E_A, v_1, v_2 to 0.

Finally, if $k_A = \widetilde{k}_A \geq 2$ but Alice has not made any transition, then she simply resets v_1, v_2 to 0.

REMARK 6.6. *If T_A and T_B have not diverged too far, then there is a common meeting point up to which T_A and T_B agree. Thus, as control information of each iteration, Alice and Bob send hash values of their two meeting points in the hope that there is a match. For a given $\widetilde{k}_A$, there are $\widetilde{k}_A$ hash comparisons that are generated. If at least a constant fraction of these comparisons result in a match, then Alice decides to rewind her transcript to the relevant meeting point. This ensures that in order for an adversary to cause Alice to backtrack incorrectly, he must corrupt the control information in a constant fraction of iterations.*

6.7 Pseudocode We are now ready to provide the pseudocode for the protocol $\Pi_{\text{enc}}^{\text{oblivious}}$, which follows the high-level description outlined in Section 6.1 and is shown in Figure 3. The pseudocode for the helper functions `AliceControlFlow`, `AliceUpdateSyncStatus`, `AliceUpdateControl`, `AliceDecodeControl`, `AliceAdvanceBlock`, `AliceUpdateEstimate`, and `AliceRollback` for Alice is also displayed. Bob's functions `BobControlFlow`, `BobUpdateSyncStatus`, `BobUpdateControl`, `BobDecodeControl`, `BobAdvanceBlock`, `BobUpdateEstimate`, and `BobRollback` are almost identical, except that "A" subscripts are replaced with "B." Also, the function `InitSharedRandomness` is the same for Alice and Bob.

References

[1] Z. Brakerski and Y. T. Kalai, *Efficient interactive coding against adversarial noise*, in 53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012, 2012, pp. 160–166.

[2] Z. Brakerski, Y. T. Kalai, and M. Naor, *Fast interactive coding against adversarial noise*, J. ACM, 61 (2014), p. 35.

[3] Z. Brakerski and M. Naor, *Fast algorithms for interactive coding*, in Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013, 2013, pp. 443–456.

[4] M. Braverman and K. Efremenko, *List and unique coding for interactive communication in the presence of adversarial noise*, in 55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014, 2014, pp. 236–245.

[5] M. Braverman and A. Rao, *Toward coding for maximum errors in interactive communication*, IEEE Transactions on Information Theory, 60 (2014), pp. 7248–7255.

[6] K. Efremenko, R. Gelles, and B. Haeupler, *Maximal noise in interactive communication over erasure channels and channels with feedback*, in Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015, 2015, pp. 11–20.

[7] M. K. Franklin, R. Gelles, R. Ostrovsky, and L. J. Schulman, *Optimal coding for streaming authentication and interactive communication*, IEEE Transactions on Information Theory, 61 (2015), pp. 133–145.

[8] R. Gelles, A. Moitra, and A. Sahai, *Efficient coding for interactive communication*, IEEE Transactions on Information Theory, 60 (2014), pp. 1899–1913.

[9] M. Ghaffari and B. Haeupler, *Optimal error rates for interactive coding II: efficiency and list decoding*, in 55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014, 2014, pp. 394–403.

[10] M. Ghaffari, B. Haeupler, and M. Sudan, *Optimal error rates for interactive coding I: adaptivity and other settings*, in Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014, 2014, pp. 794–803.

[11] B. Haeupler, *Interactive channel capacity revisited*, in 55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014, 2014, pp. 226–235.

[12] G. Kol and R. Raz, *Interactive channel capacity*, in Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013, 2013, pp. 715–724.

[13] J. Naor and M. Naor, *Small-bias probability spaces: Efficient constructions and applications*, SIAM J. Comput., 22 (1993), pp. 838–856.

[14] L. J. Schulman, *Communication on noisy channels: A coding theorem for computation*, in 33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania, USA, 24-27 October 1992, 1992, pp. 724–733.

[15] ———, *Deterministic coding for interactive communication*, in Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA, 1993, pp. 747–756.

[16] ———, *Coding for interactive communication*, IEEE Transactions on Information Theory, 42 (1996), pp. 1745–1756.

Global parameters

$b' = b + 2c\log(1/\epsilon')$	$\Pi_{\text{blk}} = B$-blocked simulating protocol for Π (see Lemma 3.1)
$N_{\mathsf{iter}} = \dfrac{n'}{b}(1 + \Theta(\epsilon\log(1/\epsilon)))$	$l' = \Theta(n\epsilon' \operatorname{polylog}(1/\epsilon'))$
$\epsilon' = \epsilon^2$	$\mathcal{C}^{\mathsf{hash}} : \{0,1\}^{\Theta(\log(1/\epsilon'))} \to \{0,1\}^{\Theta(\log(1/\epsilon'))}$ (see Section 6.5.1)
$b = s = \Theta(1/\epsilon')$	$\mathcal{C}^{\mathsf{exchange}} : \{0,1\}^{l'} \to \{0,1\}^{10\epsilon N_{\mathsf{iter}} b'}$ (see Section 6.3)
$B = sb$	$\mathcal{C}^{\mathsf{rateless}} : \{0,1\}^{B} \to \{0,1\}^{2B}$ (see Lemma 6.1)

Alice **Bob**

Random string exchange

Alice:
```
Choose a random string $\mathsf{str} \in \{0,1\}^{l'}$
$w \leftarrow \mathcal{C}^{\mathsf{exchange}}(\mathsf{str})$
```

$$w \longrightarrow \widetilde{w}$$

Bob:
```
$w' \leftarrow$ nearest codeword of $\mathcal{C}^{\mathsf{exchange}}$ to $\widetilde{w}$
$\mathsf{str} \leftarrow (\mathcal{C}^{\mathsf{exchange}})^{-1}(w')$
```

Initialization

Alice:
```
$T_A \leftarrow \emptyset$
$x \leftarrow nil$
$k_A, \widetilde{k}_A, c_A, \mathsf{sync}_A \leftarrow 1$
$E_A, v_1, v_2, j_A, \mathsf{speak}_A, a, m, \mathtt{MP1}, \mathtt{MP2} \leftarrow 0$

InitSharedRandomness()

if Alice speaks in the first block of $\Pi_{\text{blk}}$ then
   $\mathsf{speak}_A \leftarrow 1$
   $x \leftarrow$ contents of first block of $\Pi_{\text{blk}}$
   $y = y_0 \circ y_1 \circ \cdots \circ y_{2s-1} \leftarrow \mathcal{C}^{\mathsf{rateless}}(x)$
end if
```

Bob:
```
$T_B \leftarrow \emptyset$
$x \leftarrow nil$
$k_B, \widetilde{k}_B, c_B, \mathsf{sync}_B \leftarrow 1$
$E_B, v_1, v_2, j_B, \mathsf{speak}_B, a, m, \mathtt{MP1}, \mathtt{MP2} \leftarrow 0$

InitSharedRandomness()

if Bob speaks in the first block of $\Pi_{\text{blk}}$ then
   $\mathsf{speak}_B \leftarrow 1$
   $x \leftarrow$ contents of first block of $\Pi_{\text{blk}}$
   $y_0 \circ y_1 \circ \cdots \circ y_{2s-1} \leftarrow \mathcal{C}^{\mathsf{rateless}}(x)$
end if
```

Block transmission (repeat N_{iter} times)

Alice:
```
AliceUpdateControl()

Send $\mathbf{r}[i]$ in slot $z^A_{m,i}$ for $i = 1, \ldots, (b' - b)/2$
Listen during slots $\widetilde{z}^B_{m,i}$ for $i = 1, \ldots, (b' - b)/2$ and
write bits to $\widetilde{\mathbf{r}}$

if $\mathsf{sync}_A = 1$ and $\mathsf{speak}_A = 1$ then
   Send the bits of $y_{m \bmod 2s}$ in the $b$
    remaining slots
else
   Listen during the $b$ remaining slots and
    store as $g_A$
end if

AliceControlFlow()
```

Bob:
```
BobUpdateControl()

Send $\mathbf{r}[i]$ in slot $z^B_{m,i}$ for $i = 1, \ldots, (b' - b)/2$
Listen during slots $\widetilde{z}^A_{m,i}$ for $i = 1, \ldots, (b' - b)/2$ and
write bits to $\widetilde{\mathbf{r}}$

if $\mathsf{sync}_B = 1$ and $\mathsf{speak}_B = 1$ then
   Send the bits of $y_{m \bmod 2s}$ in the $b$
    remaining slots
else
   Listen during the $b$ remaining slots and
    store as $g_B$
end if

BobControlFlow()
```

End of repeat

Figure 3: Encoded protocol $\Pi_{\text{enc}}^{\text{oblivious}}$ for tolerating oblivious adversarial errors.

Algorithm 1 Procedure for Alice to process received data bits and control info from a mini-block

```
 1: function AliceControlFlow
      ▷ Update phase:
 2:   $\widetilde{\mathsf{ctrl}}_B^{(m)} \leftarrow$ AliceDecodeControl
 3:   if $\widetilde{\mathsf{ctrl}}_B^{(m)} \neq \perp$ then
 4:       $\left(\widetilde{h}_{B,c}^{(m)}, \widetilde{h}_{B,x}^{(m)}, \widetilde{h}_{B,k}^{(m)}, \widetilde{h}_{B,T}^{(m)}, \widetilde{h}_{B,\mathtt{MP1}}^{(m)}, \widetilde{h}_{B,\mathtt{MP2}}^{(m)}, \widetilde{j}_B, \widetilde{\mathsf{sync}}_B\right) \leftarrow \widetilde{\mathsf{ctrl}}_B^{(m)}$
 5:       if $\mathsf{sync}_A = 0$ then
 6:           if $\widetilde{h}_{B,k}^{(m)} \neq hash_{B,k}^{(m)}(k_A)$ or $\widetilde{\mathsf{sync}}_B = 1$ then
 7:               $E_A \leftarrow E_A + 1$
 8:           else if $hash_{B,\mathtt{MP1}}^{(m)}(T_A[1 \ldots \mathtt{MP1}]) = \widetilde{h}_{B,\mathtt{MP1}}^{(m)}$ or $hash_{B,\mathtt{MP2}}^{(m)}(T_A[1 \ldots \mathtt{MP1}]) = \widetilde{h}_{B,\mathtt{MP2}}^{(m)}$ then
 9:               $v_1 \leftarrow v_1 + 1$
10:           else if $hash_{B,\mathtt{MP1}}^{(m)}(T_A[1 \ldots \mathtt{MP2}]) = \widetilde{h}_{B,\mathtt{MP1}}^{(m)}$ or $hash_{B,\mathtt{MP2}}^{(m)}(T_A[1 \ldots \mathtt{MP2}]) = \widetilde{h}_{B,\mathtt{MP2}}^{(m)}$ then
11:               $v_2 \leftarrow v_2 + 1$
12:           end if
13:       end if
14:   else if $\mathsf{sync}_A = 0$ then
15:       $E_A \leftarrow E_A + 1$
16:   end if
17:   if $\mathsf{sync}_A = 0$ then
18:       $k_A \leftarrow k_A + 1$
19:       $\tilde{k}_A \leftarrow 2^{\lfloor \log_2 k_A \rfloor}$
20:   end if
21:   AliceUpdateSyncStatus
      ▷ Transition phase:
22:   if $k_A = \widetilde{k}_A \geq 2$ and $v_1 \geq 0.2k_A$ then
23:       AliceRollback(MP1)
24:   else if $k_A = \widetilde{k}_A \geq 2$ and $v_2 \geq 0.2k_A$ then
25:       AliceRollback(MP2)
26:   else if $k_A = \widetilde{k}_A \geq 2$ and $E_A \geq 0.2k_A$ then
27:       $a \leftarrow (m+1) \bmod 2s$
28:       $k_A, \widetilde{k}_A, \mathsf{sync}_A \leftarrow 1$
29:       $E_A, v_1, v_2, \widetilde{j}_A \leftarrow 0$
30:   else if $k_A = \widetilde{k}_A \geq 2$ then
31:       $v_1, v_2 \leftarrow 0$
32:   end if
33:   $\mathtt{MP1} \leftarrow \tilde{k}_A B \left\lfloor \frac{|T_A|}{\tilde{k}_A B} \right\rfloor$
34:   $\mathtt{MP2} \leftarrow \mathtt{MP1} - \tilde{k}_A B$
35:   $m \leftarrow m + 1$
36: end function
```

Algorithm 2 Procedure for Alice to update sync status

```
1: function AliceUpdateSyncStatus
2:     $\mathsf{sync}_A \leftarrow 0$
3:     if $k_A = 1$ then
4:         if $\widetilde{\mathsf{ctrl}}_B^{(m)} \neq \perp$ and $\widetilde{h}_{B,k}^{(m)} = hash_{B,k}^{(m)}(1)$ then
5:             if $\widetilde{\mathsf{sync}}_B = 0$ then
6:                 $\mathsf{sync}_A \leftarrow 1$; $j_A \leftarrow 0$; $a \leftarrow (m+1) \bmod 2s$
7:             else if $hash_{B,c}^{(m)}(c_A) = \widetilde{h}_{B,c}^{(m)}$ and $hash_{B,T}^{(m)}(T_A) = \widetilde{h}_{B,T}^{(m)}$ then
8:                 $\mathsf{sync}_A \leftarrow 1$
9:                 if $\mathsf{speak}_A = 0$ then
10:                    if $j_A \leq \widetilde{j}_B$ then
11:                        AliceUpdateEstimate
12:                    else
13:                        $j_A \leftarrow 0$; $a \leftarrow (m+1) \bmod 2s$
14:                    end if
15:                else
16:                    $j_A \leftarrow \min\{j_A + 1, 2s\}$
17:                end if
18:            else if $\mathsf{speak}_A = 1$ and $hash_{B,c}^{(m)}(c_A + 1) = \widetilde{h}_{B,c}^{(m)}$ and $hash_{B,T}^{(m)}(T_A \circ x) = \widetilde{h}_{B,T}^{(m)}$ then
19:                $\mathsf{sync}_A \leftarrow 1$
20:                AliceAdvanceBlock
21:            else if Bob speaks in block $(c_A - 1)$ of $\Pi_{\text{blk}}$ and $hash_{B,c}^{(m)}(c_A - 1) = \widetilde{h}_{B,c}^{(m)}$ and $hash_{B,T}^{(m)}(T_A[1 \ldots (c_A - 2)B]) = \widetilde{h}_{B,T}^{(m)}$ and $hash_{B,x}^{(m)}(T_A[((c_A - 2)B + 1) \ldots (c_A - 1)B]) = \widetilde{h}_{B,x}^{(m)}$ then
22:                $\mathsf{sync}_A \leftarrow 1$
23:                if $\mathsf{speak}_A = 0$ then
24:                    $j_A \leftarrow 0$; $a \leftarrow (m+1) \bmod 2s$
25:                else
26:                    $j_A \leftarrow \min\{j_A + 1, 2s\}$
27:                end if
28:            end if
29:        else if $\widetilde{\mathsf{ctrl}}_B^{(m)} = \perp$ then
30:            $\mathsf{sync}_A \leftarrow 1$
31:            if $\mathsf{speak}_A = 0$ then
32:                if $j_A \neq 0$ then
33:                    AliceUpdateEstimate
34:                else
35:                    $a \leftarrow (m+1) \bmod 2s$
36:                end if
37:            else
38:                $j_A \leftarrow \min\{j_A + 1, 2s\}$
39:            end if
40:        end if
41:    end if
42: end function
```

Algorithm 3 Procedure for Alice to update control information

```
1: function AliceUpdateControl
2:     $\mathsf{ctrl}_A^{(m)} \leftarrow (hash_{A,m}(c_A), hash_{A,x}^{(m)}(x), hash_{A,k}^{(m)}(k_A), hash_{A,T}^{(m)}(T_A), hash_{A,\mathtt{MP1}}^{(m)}(T_A[1 \ldots \mathtt{MP1}]), hash_{A,\mathtt{MP2}}^{(m)}(T_A[1 \ldots \mathtt{MP2}]), j_A, \mathsf{sync}_A)$
3:     $\mathbf{r} \leftarrow \mathcal{C}^{\mathsf{hash}}\left(\mathsf{ctrl}_A^{(m)} \circ hash_{A,\text{ctrl}}^{(m)}\left(\mathsf{ctrl}_A^{(m)}\right)\right) \oplus V_A^{(m)}$
4: end function
```

Algorithm 4 Procedure for Alice to decode control information sent by Bob

```
1: function AliceDecodeControl
2:     $\mathbf{z} \leftarrow$ decoding of $\widetilde{\mathbf{r}} \oplus V_B^{(m)}$ under $\mathcal{C}^{\mathsf{hash}}$ (inverse of $\mathcal{C}^{\mathsf{hash}}$ applied to nearest codeword)
3:     $\mathbf{z^c} \circ \mathbf{z^h} \leftarrow \mathbf{z}$, where $\mathbf{z^c}$ has length $(b' - b)/2$

4:     if $hash_{B,\text{ctrl}}^{(m)}(\mathbf{z^c}) = \mathbf{z^h}$ then
5:         return $\mathbf{z^c}$
6:     else
7:         return $\perp$
8:     end if
9: end function
```

Algorithm 5 Procedure for Alice to advance the block index and prepare for future transmissions

```
1: function AliceAdvanceBlock
2:     if $\mathsf{speak}_A = 1$ then
3:         $T_A \leftarrow T_A \circ x$
4:     else
5:         $T_A \leftarrow T_A \circ \widetilde{x}$
6:     end if

7:     $c_A \leftarrow c_A + 1$
8:     $j_A \leftarrow 0$

9:     if Alice speaks in block $c_A$ of $\Pi_{\text{blk}}$ then
10:        $\mathsf{speak}_A \leftarrow 1$
11:        $x \leftarrow$ contents of block $c_A$ of $\Pi_{\text{blk}}$
12:        $y = y_0 \circ y_1 \circ \cdots \circ y_{2s-1} \leftarrow \mathcal{C}^{\mathsf{rateless}}(x)$
13:    else
14:        $\mathsf{speak}_A \leftarrow 0$
15:        $a \leftarrow (m + 1) \bmod 2s$
16:        $x \leftarrow nil$
17:    end if
18: end function
```

Algorithm 6 Procedure for Alice to update her estimate of the contents of the current block based on past data blocks

```
1: function AliceUpdateEstimate
2:     $\widetilde{g}_{j_A} \leftarrow g_A$
3:     $j_A \leftarrow j_A + 1$

4:     if $j_A > s$ then
5:         $\widetilde{x} \leftarrow$ result after decoding $(\widetilde{g}_0, \widetilde{g}_1, \ldots, \widetilde{g}_{j_A - 1})$ via the nearest codeword in $\mathcal{C}^{\mathsf{rateless}}_{a, j_A}$
6:         if $hash^{(m)}_{B,x}(\widetilde{x}) = \widetilde{h}^{(m)}_{B,x}$ then
7:             AliceAdvanceBlock
8:         else if $j_A = 2s$ then
9:             $j_A \leftarrow 0$
10:            $a \leftarrow (m+1) \bmod 2s$
11:        end if
12:    end if
13: end function
```

Algorithm 7 Procedure for Alice to backtrack to a previous meeting point

```
1: function AliceRollback($\mathtt{MP}$)
2:     $T_A \leftarrow T_A[1 \ldots \mathtt{MP}]$
3:     $c_A \leftarrow \frac{\mathtt{MP}}{B} + 1$
4:     $k_A, \widetilde{k}_A, \mathsf{sync}_A \leftarrow 1$
5:     $E_A, v_1, v_2, j_A \leftarrow 0$

6:     if Alice speaks in block $c_A$ of $\Pi_{\mathrm{blk}}$ then
7:         $\mathsf{speak}_A \leftarrow 1$
8:         $x \leftarrow$ contents of block $c_A$ of $\Pi_{\mathrm{blk}}$
9:         $y = y_0 \circ y_1 \circ \cdots \circ y_{2s-1} \leftarrow \mathcal{C}^{\mathsf{rateless}}(x)$
10:    else
11:        $\mathsf{speak}_A \leftarrow 0$
12:        $a \leftarrow (m+1) \bmod 2s$
13:        $x \leftarrow nil$
14:    end if
15: end function
```

Algorithm 8 Procedure for Alice and Bob to use exchanged random string to initialize hash functions, information hiding mechanism, and encoding functions for control information

```
1: function InitSharedRandomness
2:     $p \leftarrow \Theta(\log(1/\epsilon'))$
3:     $\delta \leftarrow 2^{-\Theta(N_{\mathsf{iter}} \cdot p)}$
4:     $L \leftarrow \Theta(N_{\mathsf{iter}} n \log(1/\epsilon'))$
5:     Let $\mathsf{str} = \mathsf{str}^{\mathrm{loc}} \circ \mathsf{str}'$, where $\mathsf{str}^{\mathrm{loc}}$ is of length $\Theta(N_{\mathsf{iter}} \cdot \log^2(1/\epsilon'))$ and $\mathsf{str}'$ is of length $\Theta(\log(L/\delta))$
6:     $S \leftarrow \delta$-biased length $L$ pseudorandom string derived from $\mathsf{str}'$ (via the biased sample space of [13])
    ▷ Generate locations for information hiding in each iteration:
7:     for $i = 0$ to $N_{\mathsf{iter}} - 1$ do
8:         Choose $z^A_{i,1}, z^A_{i,2}, \ldots, z^A_{i,(b'-b)/2}, z^B_{i,1}, z^B_{i,2}, \ldots, z^B_{i,(b'-b)/2}$ to be distinct numbers in $\{1, 2, \ldots, b'\}$ using $O(\log^2(1/\epsilon'))$ fresh random bits from $\mathsf{str}^{\mathrm{loc}}$
9:     end for
    ▷ Set up parameters for encoding control information during each iteration
10:    for $i = 0$ to $N_{\mathsf{iter}} - 1$ do
11:        $V_A^{(i)} \leftarrow (b' - b)/2$ fresh random bits from $\mathsf{str}^{\mathrm{loc}}$
12:        $V_B^{(i)} \leftarrow (b' - b)/2$ fresh random bits from $\mathsf{str}^{\mathrm{loc}}$
13:        Initialize $hash^{(i)}_{A,\mathrm{ctrl}}, hash^{(i)}_{B,\mathrm{ctrl}}$ to an inner product hash function with output length $\Theta(\log(1/\epsilon'))$ and seed fixed as $\Theta(\log(1/\epsilon'))$ fresh random bits from $\mathsf{str}^{\mathrm{loc}}$
14:    end for
    ▷ Initialize hash functions for control information in each iteration:
15:    for $i = 0$ to $N_{\mathsf{iter}} - 1$ do
16:        Initialize $hash^{(i)}_{A,c}$, $hash^{(i)}_{A,x}$, $hash^{(i)}_{A,k}$, $hash^{(i)}_{A,T}$, $hash^{(i)}_{A,\mathtt{MP1}}$, $hash^{(i)}_{A,\mathtt{MP2}}$, $hash^{(i)}_{B,c}$, $hash^{(i)}_{B,x}$, $hash^{(i)}_{B,k}$, $hash^{(i)}_{B,T}$, $hash^{(i)}_{B,\mathtt{MP1}}$, $hash^{(i)}_{B,\mathtt{MP2}}$ to be inner product hash functions with output length $\Theta(\log(1/\epsilon'))$ and seed fixed using fresh random bits from $S$
17:    end for
18: end function
```