Interactive Coding with Constant Round and Communication Blowup

Klim Efremenko
Ben Gurion University, Beersheva, Israel
http://www.cs.bgu.ac.il/~klim
klimefrem@gmail.com

Elad Haramaty
Harvard University, Cambridge, MA, USA
seladh@gmail.com

Yael Tauman Kalai
Microsoft Research, Boston, MA, USA
yael@microsoft.com

Abstract

The problem of constructing error-resilient interactive protocols was introduced in the seminal works of Schulman (FOCS 1992, STOC 1993). These works show how to convert any two-party interactive protocol into one that is resilient to constant-fraction of error, while blowing up the communication by only a constant factor. Since these seminal works, there have been many followup works which improve the error rate, the communication rate, and the computational efficiency.

All these works only consider only an increase in communication complexity and did not consider an increase in *round complexity*. This work is the first one that considers the blowup of round complexity in noisy setting. While techniques from other papers can be easily adapted encode protocols with arbitrarily round complexity this coding schemes will lead to large(and usually unbounded) increase in round complexity of the protocol.

In this work, we show how to convert any protocol Π, with *no a priori* known *communication bound*, into an error-resilient protocol Π', with comparable computational efficiency, that is resilient to constant fraction of adversarial error, while blowing up both the communication complexity and the *round complexity* by at most a constant factor. We consider the model where in each round each party may send a message of *arbitrary length*, where the length of the messages and the length of the protocol may be *adaptive*, and may depend on the private inputs of the parties and on previous communication. We consider the adversarial error model, where ϵ-fraction of the communication may be corrupted, where we allow each corruption to be an *insertion* or *deletion* (in addition to toggle).

In addition, we try to minimize the blowup parameters: In particular, we construct such Π' with $(1+\tilde{O}\left(\epsilon^{1/4}\right))$ blowup in communication and $O(1)$ blowup in rounds. We also show how to reduce the blowup in rounds at the expense of increasing the blowup in communication, and construct Π' where both the blowup in rounds and communication, approaches one (i.e., no blowup) as ϵ approaches zero. We give "evidence" that our parameters are "close to" optimal.

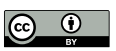
2012 ACM Subject Classification Theory of computation → Interactive computation

Keywords and phrases Interactive Coding, Round Complexity, Error Correcting Codes

Digital Object Identifier 10.4230/LIPIcs.ITCS.2020.7

Related Version A full version of the paper is available at https://eccc.weizmann.ac.il/report/2018/054/.

Funding *Klim Efremenko*: supported by the Israel Science Foundation (ISF) through grant No. 1456/18.



1 Introduction

Communication over a noisy channel is a fundamental problem in computer science, engineering and related fields. Starting from the seminal work of Shannon [30], this problem of error-resilient communication has been extensively studied. Today, we have "good" error-correcting codes – ones that achieve constant information rate as well as constant error rate. The two main error models that were considered are the *stochastic* error model, where the errors are distributed according to some distribution (such as the binary symmetric channel), and the *adversarial* error model, where errors may occur adaptively and adversarially, so long as the prescribed error rate is not exceeded. This work considers the latter (stronger) adversarial error model. In addition, we consider (adversarial) insertion and deletion errors.

In a sequence of innovative works, Schulman [27, 28, 29] initiated the study of error-resilience in the context of *interactive protocols*. Specifically, he considered the setting where two parties are interacting via a protocol over a noisy channel, where the noise could be stochastic or adversarial. Since Schulman's seminal works, there have been many followup works, that improve the error rate [9, 16, 17, 6, 1, 10], the information rate [24, 19, 12], the computational efficiency [15, 2, 4, 3], and very recently that are been beautiful works that generalize the error model of the adversary to allow insertions and deletions [8, 21, 31]. There have also been several works that consider the multi-party setting [26, 23, 7, 14]. We refer the reader to [11] for a fantastic survey on previous work on interactive coding. The focus of this work is on the 2-party setting and the adversarial error model.

All previous works consider only an increase of total communication without looking on number of rounds required to perform the task. In cryptography and in distributed computing, protocols that consist of long messages are considered, and it is desirable to keep the round complexity as low as possible. In fact, much research (in both cryptography and distributed computing) focuses on reducing the round complexity of various protocols, as often the round complexity is the bottleneck, and not the communication complexity. We argue that since we consider *interactive* protocols, we should aim for error resilient protocols, that not only blow up the communication by at most a constant factor, but also blow up the number of rounds by at most a constant factor.

Our model is the typical synchronous model used in cryptography and distributed algorithms. In our model we assume that the original protocol can be very versatile that is each party can decide how many bit he wants to send next based on previous communication and his private input. Therefore not only that number of rounds and communication of each player depends not only on the communication over the channel but also on players private input. We want to mention that more versatile original protocol is the harder it is to make a coding scheme for it. We elaborate on this model in Section 1.1.

Moreover, we emphasize that we do not assume that the communication (or round) complexity is fixed or a priori known. This is in contrast to all previous works, which assume that the communication (and round) complexity T is fixed and known in advance, and that the adversary can corrupt at most ϵT bits.[1] We note that such an assumption is often unrealistic and results in protocols where the communication complexity is always *worse-case*.

[1] We mention the work of Agrawal *et. al.* [1], which does assume that the communication complexity of the underlying errorless protocol is a priori known. However, with the goal of maximizing the error rate, the communication complexity in the error-resilient protocol is not fixed and is not a priori known. We emphasize that in our work, we do not even assume that the parties a priori know the communication complexity in the underlying errorless protocol.

In this work, we allow the communication and round complexity to differ from execution to execution, depending on the inputs, or "types" of the parties, and construct an error-resilient protocol that preserves this per-execution communication (and round) complexity. We note that the fact that we allow such adaptive (and not a priori known) communication length adds substantial technical difficulties to our work, which we elaborate on in Section 1.4.[2]

Our Results in a Nutshell. We show how to convert any protocol, where messages can be of arbitrary length, and where the communication and round complexity are not a priori known, into an error-resilient one, with comparable (computational) efficiency guarantees, that is secure against constant fraction of adversarial error, while incurring a constant blowup both to the communication complexity and to the round complexity. We allow the adversary not only to toggle with the bits of communication, but also allow the adversary to *insert* and *delete* bits. We elaborate on our communication model and error model in Section 1.1.

Moreover, we try to minimize the (constant) overhead in communication and rounds: In particular, we obtain $(1 + \tilde{O}(\epsilon^{1/4}))$ blowup in communication and $O(1)$ blowup in rounds. We also show how to reduce the blowup in rounds at the expense of slightly increasing the blowup in communication, and construct an error-resilient protocol where both the blowup in rounds and communication approaches one as ϵ approaches zero. We elaborate on our results (and on the exact parameters we obtain) in Section 1.2, we give a high-level overview of our techniques in Section 1.4, and give "evidence" that our parameters are "close to" optimal in Section 2.3 (after formally stating our main theorem in Section 2.2).

Our Technical Hurdles

The reader may at first think that dealing with short messages is the "hard case", since for long messages we can use standard error-correcting codes. We argue that this intuition is misleading. First, when considering adversarial error, applying an error-correcting code to each message separately does not help, since the entire message can be corrupted (even if the message is long), and indeed in this work we focus on adversarial error. We mention, however, that even for the case of stochastic error, dealing with messages of varying lengths, where some messages may be short while other messages may be long, is challenging.

Before explaining the difficulties that arise in this setting, we note that if we knew a priori the number of rounds and the communication complexity of the underlying protocol, then we could have "smoothed" it out perfectly, so that all the messages would have been of equal length,[3] and then we could have used a protocol (and analysis) from prior works.

Since we do not have such a bound, we cannot perfectly smooth out the underlying protocol. Nevertheless, we must somehow smooth out the protocol, since a party cannot send a long message before she is "sufficiently confident" that the transcript so far is correct, as otherwise, this long message will be wasted (even if the adversary does not corrupt it at all). Therefore, we "approximately" smoothen out the underlying protocol, by guaranteeing that each message is of length at least half and at most twice the length of the previous message. We refer the reader to Section 1.4 and Section 3 for details.

[2] We believe (though we haven't checked) that the tree-code based interactive coding schemes may easily be adapted to the setting where the communication complexity is not a priori bounded, by having each party construct an (infinitely growing) tree code. However, in tree-code based schemes the parties are computationally inefficient and there is a large blowup to the round complexity.

[3] This approach blows up the communication and round complexity by a constant factor. If the goal is to optimize this blowup (as we do in this work) then one cannot afford to perfectly smoothen out the protocol, in which case our techniques are needed.

We mention that in order to minimize the blowup, we consider two small constants $\alpha, \beta > 0$ (that depend on the error rate) and guarantee that the length of each message is at least $\alpha\ell$ and at most $\frac{\ell}{\beta}$, where ℓ is the length of the message preceding it. This is not important for the high-level overview.

We emphasize, that even after smoothing the underlying protocol, the length of the messages can still grow (or shrink) at an exponential rate, which brings rise to several challenges. For example, similar to many previous works (such as[27, 2, 19]), when a party realizes that there was an error she backtracks. In our setting we need to be extremely cautious when we backtrack. Note that the adversary can cause us to backtrack even though we are synchronized, by making us believe that we are out of sync. Previous works ensure that the adversary needs to invest enough error for such backtracking, and hence such "false" backtracking is costly for the adversary. However, in the case where messages are of varying length, this analysis becomes extremely delicate, since the adversary can corrupt a short message (by investing a small amount of his error budget), and thus falsely cause the parties to backtrack and delete a previous long message. Indeed, as opposed to previous protocols, we do not erase when we backtrack. Rather, we keep this transcript as "questionable". We refer the reader to Section 1.4 and Section 4 for details.

Moreover, when messages are of varying lengths, even if the protocol is (approximately) smooth, and even if we backtrack carefully, ensuring that the round complexity does not blow up, does not only require a careful (and significantly more complex) analysis, but also requires additional new ideas.

For example, the protocols in previous works, perform an equality test after every chunk of length d (for some parameter d), where in this equality test the parties check whether they are in sync by sending each other a hash of their transcript so far. In our setting, messages may be very long, and we cannot chop a message to chunks of d bits each, since this will blow up the round complexity. Instead, it is tempting to simply append to each message a hash of length that is proportional to the message length (e.g., append a hash of length $\lfloor \frac{\ell}{d} \rfloor$ to a message of length ℓ). However, as we show in Section 1.4 and Section 5, in order to ensure a constant blowup in round complexity, we must not only allow the length of the hash value to depend on the length of the message it is being appended to, but rather it should also depend on the length of the *entire history*. This is the case since if the protocol has messages of varying lengths, the adversary can corrupt a single long message, in a way that causes many hash collisions in future short messages. Thus, by corrupting one (long) message many rounds can be wasted.

In order to get around this problem, we allow the length of the hash to depend on the length of the entire history. Moreover, we consider randomized (i.e., seeded) hash function, where the party sends the hash value together with the hash seed, so that the adversary does not know which hash function will be used ahead of time. However, with a seed of length w one can hash messages of length at most 2^w, and the history may be longer than 2^w. Thus, in our scheme some of the seed is chosen ahead of time and some of the seed is chosen with each message. We refer the reader to Section 1.4 and Section 5 for details.

Moreover, the fact that the communication complexity is not a priori known creates an additional problem. Following previous works (such as [2, 3, 19]), we first construct a protocol in the *common random string* (CRS) model (this is done in Section 5), and then we remove the CRS (in Section 6). Removing the CRS in previous work was straightforward: First show that the CRS can be made relatively short (of size proportional to the communication complexity) by using a δ-biased source, and then argue that one of the parties can simply send the CRS using a (standard) error correcting code. In our case this cannot be done since we do not have an a priori bound on the communication complexity.

We give an overview on how we overcome the technical hurdles mentioned above in Section 1.4, but warn the reader that overcoming these challenges is quite difficult, and results in a very complex analysis.

We next explain our model in more detail.

1.1 Our Model

1.1.1 The Noiseless Model

We consider 2-party protocols, between two parties, Alice and Bob. In our model, at every round i, Alice and Bob do the following: Alice chooses $\ell_A(i) \in \mathbb{N}$ (greater than 0) and a message $m_A(i) \in \{0,1\}^{\ell_A(i)}$, based on her view of previous communication and her private input, and sends $m_A(i)$ to Bob. Similarly, Bob chooses $\ell_B(i) \in \mathbb{N}$ and a message $m_B(i) \in \{0,1\}^{\ell_B(i)}$, based on his view of previous communication and his private input, and sends $m_B(i)$ to Alice. At some round, one of the parties aborts, and both parties report an output.

More generally, we allow Bob's message in the i'th round to depend, not only on all previous communication and his private input, but also on Alice's message in the i'th round. This corresponds to the synchronous model where in each round i, Alice and Bob do not send their messages simultaneously, but rather first Alice sends her message and only then Bob sends his message (which may depend on Alice's message). This model is known as the message-passing model, and is the most common model used in cryptography (and distributed algorithms). We note that our results also apply to the synchronous simultaneous message model, and the choice of presenting our results in the synchronous message-passing model was due to the fact that we think that this model is more standard.

We denote the input of Alice by x, and we denote the input of Bob by y. Note that a pair of inputs (x, y) define ℓ_A and ℓ_B for all rounds, and also define the number of rounds. Thus, in the noiseless setting, for any protocol we can define $\mathrm{CC}(x, y)$, which is the communication complexity of the protocol for the input pair (x, y). Similarly, we can define $\mathrm{R}(x, y)$, which is the number of rounds for the input pair (x, y).

1.1.2 The Noisy Model

In this work, we consider the adversarial error model, and assume that the adversary can corrupt any ϵ-fraction of the bits, for some a priori fixed small constant $\epsilon > 0$. We allow the adversary, not only to toggle with the bits, but he can also insert and delete bits.

In our model, where messages can be of arbitrary length, protecting protocols against insertions and deletions is extremely important, since otherwise the parties can securely encode information via the *length* of the messages. Specifically, in our model, where messages can be of varying lengths, one can trivially protect protocols against (adversarial) toggle corruptions while incurring only a constant factor blowup in the communication complexity, albeit an *unbounded blowup* in the round complexity, as follows: First convert the protocol to a protocol where each party sends a single bit in each round. Then, encode this bit as follows: If the bit is zero then encode it via a single bit (zero or one), and if the bit is 1 then encode it via any two bits. Upon receiving an encoded message the parties will decode without looking at the content of the message, but rather only by the length of the message.

We note that most previous work on interactive coding do not consider insertion and deletions. Indeed, in the synchronous model, where the parties send one bit per round, insertions and deletions are not interesting, since the parties "can tell" when an insertion or deletion occurs.

An exception are the recent works of [8, 21, 31]. These works consider insertion and deletions in the *asynchronous* model. More specifically, they consider an adversary who can insert a message from one party and delete a message from the other party, and thus cause the parties to be out-of-synch with regard to which round they are on (though similarly to previous work, they are in the bit-by-bit model, thus their protocols incur a large blowup to the round complexity, and they assume an a priori bound on the communication complexity).

In our work, we consider the *synchronous* model, where the parties always agree on the number of speaking alternations (which in our case is exactly the number of rounds). We emphasize, however, that the work of [21] shows a generic method for converting any error-resilient protocol in the synchronized model into one that that is error resilient in the asynchronized model. We believe that one can use their approach (and in particular the use of a synchronization code [20]) to boost our result from the synchronous model to the asynchronous model.

In the adversarial error model, in our work and in all previous works, there is an a priori fixed constant ϵ and it is assumed that the adversary can corrupt at most ϵT bits, where T is the number of bits communicated. In most previous work, the value of T was assumed to be a priori known (and fixed). As mentioned above, in this work, we allow the length of the protocol to depend on previous communication. This models the real world setting, where we cannot a priori predict the length of our conversations, and it can depend on our private inputs (or on our "types").

In this model, care should be taken when defining the adversarial error model. One possibility is to allow the adversary to corrupt ϵT bits, where T is the number of bits that would have been transmitted assuming no error.[4]

We note, however, that with such a definition the adversary can use his ϵT bits of corruption budget, and cause the parties to abort prematurely. Namely, he can convince both parties that the other party is "boring" (i.e., that the other party has an input such that if they were executing the error-free protocol without error, the number of bits exchanged would have been less than ϵT). In such case both parties would abort prematurely and the adversary would "win".

Instead, we allow the adversary to corrupt only ϵ-fraction of the bits that were actually communicated. We note that a similar model was used in the work of Agrawal et al. [1], where their goal was to get optimal error-rate, and to that end, they considered error-resilient protocols with an adaptive speaking order and where the communication complexity may depend on the error pattern.[5] We emphasize that this adversarial model is stronger than alternative (natural) models, such as the the prefix model that allows the adversary to corrupt ϵ-fraction of any prefix of the transcript.

In this work, we also add a bound ϵ' on the number of rounds that the adversary can "fully" corrupt, where we say that a round is *fully corrupt* if the adversary corrupts more than δ-fraction of the bits, for some small constant δ (which depends on the error bound ϵ). We note that all previous works also had such a bound (implicitly), since in previous works there was no distinction between rounds and communication. In contrast, in our model, a bound on the number of bits corrupted does not imply a bound on the number of rounds that are fully corrupted. For example, consider the protocol in which there is one long message of length ℓ, followed by $\epsilon\ell$ messages, each consisting of a single bit. In such a protocol, not corrupting the long message gives the adversary the budget to corrupt all the short messages.

[4] We note that the adversary knows T since we assume (similarly to all previous work that consider the adversarial error model), that the adversary knows the private inputs of the parties.

[5] As mentioned above, the work of [1] does assume an a priori known bound on the communication complexity of the underlying (errorless) protocol.

We emphasize that bounding the number of rounds that are fully corrupted is necessary, since without such a bound, it is impossible to ensure a small blowup in round complexity. This argument is deferred to Section 2.3 where we give evidence to the optimality of our parameters.

1.2 Our Results

In what follows, we denote our error parameters by ϵ and ϵ', where ϵ corresponds to the fraction of corrupted bits, and ϵ' corresponds to the fraction of (fully) corrupted messages. We show that for any (small enough) constants $\epsilon, \epsilon' > 0$ there exist blowup parameters $\alpha, \alpha' > 0$ such that one can convert any protocol into an error resilient one (with respect to ϵ and ϵ'), with α blowup in communication, and essentially α' blowup in rounds (with an additional term that depends logarithmically on the communication complexity). We can set $\alpha' = O(1)$ we obtain a blowup of $\alpha = \tilde{O}\left(\epsilon^{1/4}\right)$ in communication complexity. Alternatively, one can set α, α' such that they both approach 0 as ϵ, ϵ' approach 0.

Our error-resilient protocol is randomized, even if the original protocol was deterministic. This is similar to all previous works that construct computationally efficient interactive coding schemes that are robust to adversarial error (starting with the work of [2]). Schulman [28] (followed by many followup works) gave a deterministic interactive coding scheme, at the price of computational inefficiency. The parties in the error resilient scheme run in exponential time in T, where T is an upper bound on the length of the underlying protocol.[6] Recently, Gelles et al. [13] gave a deterministic and efficient construction for the case of random error. However, constructing a deterministic interactive coding scheme that is resilient to adversarial error and is computationally efficient remains an interesting open problem.

We are now ready to state our main theorem. The most general theorem can be found in Section 2, and in what follows we present our theorem in a regime of parameters that we think is of particular interest.

In what follows, we let $t_{\min}$ denote the minimum value for which the underlying error-free protocol Π transmits at least $t_{\min}$ bits.

▶ **Theorem 1** (Main Theorem (informal)). *For any sufficiently small $\epsilon \geq 0$ and for any $\epsilon' \leq \epsilon^{1/4}$, there exist blowup parameters α and α', and a polynomial time probabilistic oracle machine S, such that the following holds. For any adversary $\mathcal{A}$ that corrupts at most ϵ-fraction of the bits of the simulated protocol $\Pi'_{\mathcal{A}}$ (which is the protocol Π' executed with the adversary $\mathcal{A}$), and "fully" corrupts at most ϵ'-fraction of the messages of $\Pi'_{\mathcal{A}}$, where $\mathcal{A}$ "fully" corrupt a message if he corrupts at least α^2-fraction of the bits of the message, we have the following guarantees.*

1. $\mathrm{CC}(\Pi'_{\mathcal{A}}) \geq t_{\min}$.
2. $\Pr\left[\mathrm{CC}(\Pi'_{\mathcal{A}}) > (1+\alpha)\mathrm{CC}(\Pi)\right] = \exp\left(-\mathrm{CC}(\Pi'_{\mathcal{A}})\right)$.
3. $\Pr\left[R(\Pi'_{\mathcal{A}}) > (1+\alpha')R(\Pi) + \alpha' \log \mathrm{CC}(\Pi)\right] = \exp\left(-R(\Pi'_{\mathcal{A}})\right)$.
4. $\Pr\left[(\mathsf{Output}(\Pi'_{\mathcal{A}}) \neq \mathsf{Trans}(\Pi))\right] = \exp\left(-\mathrm{CC}(\Pi'_{\mathcal{A}})\right)$.

Moreover, we can choose the parameter α, α' such that $\alpha = \tilde{O}(\epsilon^{1/4})$ and $\alpha' = O(1)$, or we can choose α, α' such that α and α' approach 0 as ϵ approaches 0.

The purpose of $t_{\min}$. In the theorem above, without adding the restriction that $CC(S^A, S^B) \geq t_{\min}$, the simulated protocol could have aborted as soon as more than ϵ-fraction of error was detected. In particular, if the first bit was noticeably corrupted, then the parties in

[6] Braverman [5] showed how to improve the parties' runtime to be sub-exponential in T.

the simulated protocol could have safely aborted. Thus, without adding the restriction that $CC(S^A, S^B) \geq t_{\min}$, this theorem does not even generalize previous works, which all assume an a priori fixed transcript size t and assume the adversary makes at most ϵt corruptions. Adding this restriction, gives the adversary an initial budget of $\epsilon t_{\min}$ corruption bits. Moreover, since the error probability is exponentially small in the actual transcript length, the requirement $CC(S^A, S^B) \geq t_{\min}$ guarantees a low error probability.

There was a long line of works that consider the case of unknown noise of

1.3 Related Works

This work is the first one that considers the blowup of round complexity in noisy setting. While techniques from other papers can be easily adapted encode protocols with arbitrarily round complexity this coding schemes will lead to large(and usually unbounded) increase in round complexity of the protocol. However there are many papers that considered models related to our model. We want to mention that in other papers consider an effect of message length(what mean also the round complexity) on the rate of the communication complexity of the coding one can see it non-explicitly in [24, 19] and more explicitly in [22]. In non-noisy setting there was a long line of works showing that reducing round complexity may cause an exponential gap in communication complexity.

The paper [1] considered a very adaptive models of the protocols where total communication is not known in advance. There was also a large line of works considering the case when to total amount of noisy in unknown see survey by Gelles [11] for more details.

1.4 Overview of Our Techniques

In this section we give a high-level overview of the main ideas behind our construction and our analysis. In this overview, we do not focus on getting "optimal" parameters, and focus on constructing an error-resilient scheme that blows up the round and communication complexity by a constant factor. We note that all the conceptual ideas in this work are needed even to achieve constant overhead.

We start with an arbitrary protocol Π.

Smoothness. We first convert Π into a smooth protocol, with the property that after a message of length ℓ comes a message of length at most 2ℓ, and before a message of length ℓ comes a message of length at least $\ell/2$. We mention that in the actual protocol, to minimize the blowup in rounds and communication, we define (α, β)-smoothness, where α and β are functions of the error rate ϵ, and the guarantee is that after a message of length ℓ comes a message of length at least $\alpha\ell$ and at most $\frac{\ell}{\beta}$, and we show how to convert any protocol Π into an (α, β)-smooth protocol.

As mentioned above, the reason we need to smoothen Π is that otherwise, if after receiving a short message a party sends a long message, then the adversary by corrupting the short message, can cause the long message to be wasted, thus effectively allowing him to corrupt the long message by only using the budget needed to corrupt the short message.

Intuitively, we smoothen Π by instructing a party who wishes to send a long message after receiving a short message, to do so "cautiously", by sending the long message over several rounds, each time increasing the message length by at most a factor of 2.

To ensure that this does not cause a blowup to the round complexity, we make sure that a party does not send a short message after receiving a long one. Otherwise, suppose Alice always sends long messages (each of length ℓ) and Bob always sends single bit messages. Then

by having Alice send her messages "cautiously", as explained above, the round complexity will blowup by a factor of $\log \ell$, which is too large. Instead, we instruct Bob to send longer messages, of length $\alpha\ell$, so that the adversary will need to invest enough budget to corrupt Bob's message; in particular, enough to allow Alice to send her length ℓ message safely.

We refer the reader to Section 3 for the formal definition of smoothness, and to Lemma 6 for how to convert a protocol into a smooth one.

From now on we assume the protocol Π is smooth, and show how to convert it into an error resilient one.

Message adversary. We first note that we can focus our attention only on adversaries, that rather than corrupting individual bits, corrupt messages, where the price of corrupting a message m is the maximum between the length of m and the length of the corrupted version of m. If the adversary chooses to corrupt a message m then he may corrupt it adversarially, and if the adversary chooses not to corrupt a message then he cannot make any changes to it.

The reason we can focus on such adversaries is that we can easily convert any protocol that is resilient to errors made by message adversaries into a protocol that is error resilient to any adversary by applying an error correcting code to each message, and hence if only a small fraction of a message is corrupted (smaller than the allowed error rate) then this corruption can be ignored, since it is immediately corrected by the error correcting code. We use the error correcting code of Guruswami and Li [18], that is resilient to insertion and deletions, and has a minimal blowup of $1 + \tilde{O}(\sqrt{\epsilon})$ to the message length.

Thus, from now on, throughout this section, we ignore the layer of error correcting code, and consider only message adversaries.

1.4.1 The Protocol in the Ideal Hash Model

We first show how to convert any protocol Π into a protocol that is error resilient in the *Ideal Hash Model*. As in previous works (starting with the original work of [27]), our starting point is the idea of using hashing to check for consistency. Namely, in the protocol Alice and Bob check equality of their partial transcripts, by sending to each other hashes of their partial transcripts.

In the Ideal Hash Model, we assume the existence of an "ideal" hash function, that is known to all parties and does not need to be communicated, and in the analysis we assume that the number of hash collisions is bounded, yet adversarially chosen (where the cost for each hash collision is proportional to the length of the hash value). We later elaborate on how we remove this ideal model assumption, by implementing this ideal hash using a real hash function.

For the sake of simplicity, throughout this overview we think of the parties appending to each message they send a hash of their transcript so far. We mention however, that in the actual protocol, since we want to optimize the communication blowup, we append a hash only to "long enough" messages, i.e., messages of length at least d, for a carefully chosen parameter $d \in \mathbb{N}$. In particular, we do not append a hash to short messages, and instead add a hash in every round that divides d (to take care of the case where all the messages are short).

Each party, upon receiving a message, first checks the consistency of the corresponding hash with its current transcript. If an inconsistency occurs, the parties enter a *correction mode*.

Correction Mode. In correction mode, the parties realize that their transcripts are inconsistent, and they need to rewind their transcript to a point where they believe they are consistent, yet without backtracking too much. Note that once an error is detected, the parties cannot simply rewind their transcript one round at a time, since the adversary can cause them to completely get out of sync. Moreover, they cannot send each other the round number they are currently simulating, as was done in [2], since this will blowup the communication by too much. Instead, we adapt the idea of backtracking to a "meeting point", an idea that was originated in [27] and used in [19]. For the sake of completeness, we explain this idea below.

Once the parties realize they are not in sync, they enter a correction mode, and once in error mode, they send two hashes of their transcript: One hash of the entire transcript, and the other of the transcript up until the second largest round. If a consistency was found they go back to the point of consistency. Otherwise, they send two hashes of their transcript until the largest, and second largest, round which is a multiple of 2. Again, if a consistency was found they go back to the point of consistency. Otherwise, in the i'th try, they send two hashes of their transcript until the largest, and second largest, round which is a multiple of 2^{i-1}.

In order to avoid the situation where the adversary invests $O(1)$ corruptions, and causes a party to go back 2^i steps, and thus lose 2^i bits of a possibly good transcript, the parties go back 2^i steps only after receiving roughly 2^i confirmations. The confirmations cannot be in a single round, since then the adversary could corrupt a single round and cause the parties to go back (possibly) 2^i rounds. Thus, instead these confirmations should span roughly 2^i rounds, and each party keeps a counter of how many confirmations it has.

One important missing piece is that they can be out of sync with respect to which are the meeting points. Thus, we also append to the message a hash of E, which denotes the number of rounds the party is in the error mode, and this length determines where the meeting points should be (which is roughly the power of 2 closest to E).

In previous works, once the parties backtrack, they erase the (seemingly) inconsistent transcript and continue to simulate the actual protocol. One important point where our protocol differs from all previous work, is that in our protocol the parties cannot afford to erase their (seemingly) inconsistent transcripts. This is due to the fact that the messages in the (seemingly) inconsistent part may be very long. For example, consider the case where the last message added to the transcript is of length 1, the one prior is of length 2, the one prior is of length 4, then length 8, and so on. Suppose no errors occurred and everything is consistent. The adversary can corrupt the hash appended to the short (1 bit) message, making the parties believe that their transcripts are inconsistent. The parties will backtrack, but the adversary will continue to make them believe that they are inconsistent, so that they erase i messages. This means erasing 2^i bits of communication, which is way more than the parties can afford to erase. Therefore, in our protocol, rather than erasing the (seemingly) inconsistent transcript, we keep it as questionable, and enter what we call a *verification mode*.

Verification Mode. In the verification mode, the parties simply test whether their questionable messages are consistent. They do this round-by-round, by sending a hash of the messages corresponding to each round. If their hashes agree, they mark the round as valid, and continue to the next round. If they arrive to a round where their messages do not agree, they don't immediately erase all the questionable transcript. Rather, they erase it only after they are "sufficiently" confident that they are inconsistent. To this end, they

send longer and longer hashes until the number of bits of hash are proportional to the (seemingly) inconsistent transcript, and if the inconsistency persists then the parties erase their questionable transcript, and continue to simulate the underlying protocol.

This protocol is formally presented in Section 4, and the formal analysis in the Ideal Hash Model can be found in Section 4.2 and proof will appear in full version.

1.4.2 Our Protocol in the Shared Randomness Model

We next show how to implement the ideal hash functions with a specific hash function. To this end, we construct a function family $\mathcal{H} = \{h_x\}$, where each hash function h_x is associated with a (possibly long) seed x.

We consider the *shared randomness model*, where the parties are allowed to share a (possibly long) random string. We later show how to eliminate the need for shared randomness. But for now, we assume that the shared randomness is as long as we need. In particular, we use a different hash function (i.e, a different seed) for each equality test, and assume that the shared random string contains all these seeds. Since the length of the protocol is adaptive and not a priori bounded, the length of the common random string is also not a priori bounded.[7]

We emphasize that the shared randomness (and in particular the seeds) are known to the adversary. Therefore the adversary, given a seed x, can try to skew the protocol and cause the parties to send many messages whose hashes collide.

Note that the adversary has $\binom{t}{\epsilon t} = 2^{\tilde{O}(\epsilon)t}$ different ways to corrupt the t bits of the communication. Thus, he can cause hash collisions in approximately $\tilde{O}(\epsilon)t$ bits. If we append each message of size ℓ with $O(\ell)$ bits of hash, the adversary will be able to cause hash collisions in messages with total volume of $\tilde{O}(\epsilon)t$, which is within the allowed error range. Indeed, our main challenge is to bound the number of *rounds* with hash collisions, a challenge that previous works did not need to deal with since in their setting, communication complexity and round complexity are equivalent.

If we a priori knew the length of the transcript t and the number of rounds R, then we could add $U = \frac{t}{R}$ bits of hash to each message, and since the adversary can cause only $\tilde{O}(\epsilon)t$ bits of hash collisions, the number of rounds in which the adversary can cause a hash collision, is bounded by $\tilde{O}(\epsilon)R$, which is again within our allowed error range.

Since we don't have such a bound, it is tempting to append to each message sent in round r a hash of length $U_r = \frac{t_r}{r}$, where t_r is the communication up to the round r. But the following example shows that such a padding does not suffice, and the adversary can still force too many rounds with hash collisions.

Consider a protocol that consists of $\tilde{O}(1/\epsilon)$ chunks such that chunk 0 consists of k single bit messages, and each chunk $i \neq 0$ consists of a single (long) message of length $2^i k$, followed by $\tilde{O}(\epsilon)k$ single bit messages. Note that in this case, the total number of hash bits in chunk i is $\approx 2^i$, and thus an adversary that corrupts the long message of this chunk can cause hash collisions in all the rounds of the chunk, resulting with a total of $O(R)$ rounds with hash collisions.[8]

[7] We later show how we convert any such protocol in the *unbounded* shared randomness model into one that uses only private randomness.

[8] to be more precise, we need a long enough message at the end of the protocol to give the adversary enough budget to corrupt all of the long messages.

To overcome this issue, the idea is to partition the protocol to chunks (which we call regimes), and append to each message a hash of length that is proportional to the average length of a message in the chunk. To be precise, we append to each message a hash of length $U_r = \max_{r'<r} \frac{t_r - t_{r'}}{r - r'}$. Unfortunately, in this case the total amount of hash bits being added can be as large as $t \log t$, which we cannot afford.

To overcome this issue, in each round, instead of sending all the U_r bits of hash, we send only a hash of these bits, where the seed of this (outer) hash is chosen using private randomness. Specifically, rather than sending $H_x(T)$ (which consists of U_r bits), the party chooses a random seed S, and sends $H_S(H_x(T))$, together with S. This reduces the number of bits being communicated from U_r to $\log U_r$. Since the adversary does not a priori know the private randomness chosen by the parties, he cannot corrupt the history to cause a hash collision in H_S in too many rounds. Moreover, since we saw that he cannot cause hash collisions in $H_x(T)$ in too many rounds, these hash functions are "safe". We note that a similar idea of using a randomized hash function was used by Haeupler [19], for the sake of improving the rate of his interactive coding scheme. We refer the reader to Section 5 formal description of the hash function.

Finally, to conclude the analysis, we need to show that adding these (randomized) hash functions does not blow up the communication by too much. More precisely, one needs to show that $\sum_r \log U_r = O(t)$. This analysis is extremely delicate and requires several new ideas.

1.4.3 The Protocol in the Private Randomness Model

Finally, we show how to remove the need for shared randomness, while using only the private randomness of the parties. Namely, we show how to convert any protocol Π in the shared random string model, to one that uses only private randomness.

The basic idea is to follow the approach used in previous works (such as [2, 19]), and replace the long shared randomness with $2^{-O(T)}$-biased randomness, where T is an upper bound on the communication complexity. Such $2^{-O(T)}$-biased randomness can be generated using only $O(T)$ random bits. So, the basic idea is to send these $O(T)$ bits of randomness in advance, using an error correcting code. If we indeed had a bound T on the communication complexity, then this idea would work, and we would be done.

However, in our setting, we do not have an a priori bound on the communication complexity. In particular, if the communication complexity exceeds $O(T)$, then the adversary has the budget to corrupt more than $O(T)$ bits, and hence can completely corrupt the randomness s. We overcome this problem by sending more (and "safer") randomness as the communication complexity increases.

The protocol starts when one of the paries, say Alice, samples the shared random string $s_1 \in \{0,1\}^{O(t_{\min})}$ on her own (using her private randomness), and sends it to Bob. Then the parties execute Π with s_1 as the shared randomness. Once the communication complexity exceeds $O(t_{\min}/\epsilon)$, where ϵ is the corruption rate of the adversary, the random string s_1 is no longer "safe", and the the parties exchange a new random string s_2 of length $O(t_1)$, where t_1 is the current communication complexity. In addition to sending the new random string s_2 the parties also resend the previous random string s_1. The reason for resending previous seeds is that by resending the seeds the goal is to ensure that if one of the seeds was ever corrupted then the parties will "catch" the adversary, since the adversary does not have enough budget to continue to corrupt that seed, and the first time that he does not corrupt it, the parties will notice the inconsistency and abort, with the guarantee that the adversary performed too many errors.

In a similar way, after the communication complexity exceed $t_2 = O(t_1/\epsilon)$ a party will choose at random s_3 such that $|s_1| + |s_2| + |s_3| = t_2$, and will send (s_1, s_2, s_3), where t_2 is the current communication complexity, etc. As mentioned above, we ensure that if at any point, a message encoding randomness was decoded incorrectly, then eventually the paries will abort, and "catch" the adversary with injecting too many errors. This guarantee simplifies the analysis: Either at some point a randomness message was decoded incorrectly, in which case the adversary is "caught" with injecting too many errors, or all the parties always agree on the randomness, in which case correctness follows from the correctness of the underlying protocol in the shared randomness model.

A minor problem with the above idea is the following: a randomness message (s_1, s_2) may have been corrupted and converted into a protocol message, and a few rounds later a protocol message could have been corrupted and converted into the same randomness (s_1, s_2). To ensure that the parties will notice such corruption, we add to the randomness also the rounds $r_1, \ldots, r_k$ in which randomness were sent.

However, there is still a problem with the above idea, which is that in the early stage of the protocol, the shared random string has relatively large bias since it is generated using a short seed, and yet the adversary may have the budget to corrupt many bits, since the total communication may be large. It can be shown that such a powerful adversary can make too many hash collisions in the first part of the protocol.

To overcome this problem, we "enforce" that the adversary corrupts at most $O(\epsilon)$ fraction of any prefix of the protocol. To do so, in each time t_i, in addition to sending $(s_1, ..., s_{i+1})$ (together with $(r_1, \ldots, r_i)$), the parties send the transcript they have seen so far. If the parties detect that the adversary made significantly more than the allowed ϵ fraction of error, they abort, causing him to fail by exceeding his allotted corruption budget.

The formal protocol is described in Section 6.

2 Our Results

In this section we present our main theorem, and give an intuitive argument for why our parameters seem to be optimal. We start by introducing notations and definitions that we use in our theorem, and throughout the manuscript.

2.1 Notations and Definitions

For any 2-party protocol $\Pi = (A, B)$, we denote by $\mathsf{Trans}(\Pi)$ the transcript of Π, which consists of all the messages exchanged throughout an execution of the protocol Π. We denote by $\mathsf{Output}(\Pi)$ the output of the parties after executing Π. We think of the protocol Π as being a deterministic protocol with no inputs. This is without loss of generality since we can always hard-wire the randomness and input into the protocol. We denote by $\mathrm{CC}(\Pi)$ the communication complexity of Π, and we denote by $R(\Pi)$ its communication complexity.

We consider simulators for simulating an interactive protocols. A simulator is a probabilistic oracle machine, that uses a protocol $\Pi = (A, B)$ as an oracle, and produces a new protocol $\Pi' = (S^A, S^B)$ that outputs the transcript of Π (even in the presence of error). For any adversary $\mathcal{A}$ we denote by $\Pi'_{\mathcal{A}}$ the protocol Π' executed with the adversary $\mathcal{A}$.

▶ **Definition 2.** *We say that an adversary $\mathcal{A}$ corrupts at most ϵ-fraction of the bits of a protocol Π' if the number of corruptions made by $\mathcal{A}$ is at most $\epsilon \mathrm{CC}(\Pi'_{\mathcal{A}})$, where each corruption is either a toggle, an insertion or a deletion. The adversary $\mathcal{A}$ can be computationally unbounded, and its corruptions may depend arbitrarily on states of both parties in Π'.*

▶ **Definition 3.** *We say that a message is γ-corrupted if the adversary corrupts at least γ-fraction of the bits of the message.*

We say that $f(x) = \tilde{O}(g(x))$ if there exists a $c \in \mathbb{N}$ such that $f(x) = O\left(g(x)\log^c\left(g(x)\right)\right)$ and we say that $f(x) = \tilde{\Omega}(g(x))$ if there exists $c \in \mathbb{N}$ such that $f(x) = \Omega\left(g(x)\log^{-c}\left(g(x)\right)\right)$.

2.2 Our Main Theorem

▶ **Theorem 4.** *There exists a universal constant $\alpha_0 \geq 0$ such that for any blowup parameters $\alpha \leq \alpha_0$ and $\alpha' \leq 1$, there exist parameters $\epsilon = \tilde{\Omega}\left(\alpha^{3+\frac{1}{\alpha'}}\right)$, $\epsilon' = \tilde{\Omega}(\alpha{\alpha'}^3)$, and $\delta = \alpha^{O(1/\alpha')}$, and there exists a probabilistic oracle machine S, such that for any protocol $\Pi = (A, B)$, in which the parties always transmit at least $t_{\min}$ bits (even in the presence of error), and for any adversary $\mathcal{A}$ that corrupts at most ϵ-fraction of the bits of the simulated protocol $\Pi'_{\mathcal{A}}$, the protocol $\Pi'_{\mathcal{A}}$ (which is the protocol Π' executed with the adversary $\mathcal{A}$), satisfies the following properties.*

1. $\mathrm{CC}(\Pi'_{\mathcal{A}}) \geq t_{\min}$.
2. *There exists $t_0 = (1 + \tilde{O}(\alpha))\mathrm{CC}(A, B)$ such that for all $t > t_0$*

$$\Pr[\mathrm{CC}(\Pi'_{\mathcal{A}}) > t] \leq 2 \cdot 2^{-\delta t} ,$$

where the probability over the private randomness of S.

3. *There exists $r_0 = (1 + O(\alpha'))\, R(A, B) + O\left(\frac{1}{\log\frac{2}{\alpha'}}\log \mathrm{CC}(A, B) + 1\right)$ such that for any $r \geq r_0$, if at most ϵ'-fraction of the messages are α^2-corrupted, then*

$$\Pr\left[R(\Pi'_{\mathcal{A}}) > r\right] \leq 2 \cdot 2^{-\delta r} ,$$

where the probability over the private randomness of S.

4. *For any $t > 0$,*

$$\Pr\left[(\mathsf{Output}(\Pi'_{\mathcal{A}}) \neq \mathsf{Trans}(\Pi)) \ \wedge \ (\mathrm{CC}(\Pi'_{\mathcal{A}}) > t)\right] \leq 2 \cdot 2^{-\delta t} ,$$

where the probability over the private randomness of S.

5. *S is a probabilistic polynomial time oracle machine, and hence the computational efficiency of S^A and S^B is comparable to that of A and B, respectively.*

In Section 2.3 below, we give an intuitive argument for why our parameters seem to be optimal. Then, the rest of the manuscript is devoted to proving Theorem 4. Before, explaining our choice of parameters, in what follows, we give a high-level overview of the structure of the proof of Theorem 4.

Road Map. We first convert $\Pi = (A, B)$ into a smooth protocol Π_{smooth}. We show how this can be done in Section 3. Then, in Section 4, we show how to convert any smooth protocol Π_{smooth} into a protocol Π_{ideal}, which is error-resilient in the ideal hash model. In this model, we assume that the adversary is a "message adversary", which means that if he corrupts even a single bit of a message the price he pays for such a corruption is the length of the entire message (more precisely, the maximum between the length of the original message and the length of the corrupted version of it). Moreover, we assume that the number of hash collisions is bounded and adversarially chosen. We refer the reader to Section 4 for details.

In Section 5, we show how to convert Π_{ideal} into a protocol Π_{rand_1}, which is error resilient in the common random string model, assuming the adversary is a "message adversary". Loosely speaking, this is done by instantiating the ideal hash using public (and private)

randomness. In Section 6, we show how to instantiate the common random string using private randomness, to obtain a protocol Π_{rand_2} that is error resilient against any "message adversary". Finally, we convert Π_{rand_2} into $\Pi' = (S^A, S^B)$, where Π' is the same as Π_{rand_2}, except that each message is sent encoded with the error correcting code that is resilient to insertions and deletions. In Section 7, we "put it all together" and prove that Π' is the error resilient protocol guaranteed in Theorem 4 above.

2.3 Intuition Behind our Parameters

In what follows, we give an intuitive argument for why our parameters seem to be optimal. We emphasize that this is by no means a proof of optimality, but rather an intuition for where these parameters came from.

As mentioned above, since messages can be of arbitrary length, and since we do not want to blow up the round complexity by much, we must use an error-correcting code that is resilient to (adversarial) insertions and deletions. To date, the maximal rate error-correcting code that is resilient to (adversarial) insertions and deletions is due to Guruswami and Li [18]. This code blows up the message length by $1 + \tilde{O}(\sqrt{\epsilon})$ and is resilient to ϵ fraction of errors.

Moreover, as argued in Section 1.4, in order to ensure a small blowup in communication our error-resilient protocol must be relatively "smooth". In other words, in the error-resilient protocol, after a message of length ℓ we should not send a message much longer than ℓ, since then the adversary will corrupt the length ℓ message and as a result will cause the next long message to be obsolete. Suppose for simplicity (for now) that all the messages are all of the same length ℓ.

Suppose our protocol has blowup $1 + \tilde{O}(\alpha)$ in communication complexity. Thus, we can use the error-correcting code of [18] that blows up the message length by at most $(1 + \tilde{O}(\alpha))$. This code is resilient to α^2 fraction of errors. Thus, by corrupting $\alpha^2\ell$ bits of a message the adversary can make the next round completely obsolete. Since the adversary can corrupt ϵ-fraction of the bits, he can make $\frac{\epsilon}{\alpha^2}$-fraction of the rounds obsolete, which implies that it must be the case that $\frac{\epsilon}{\alpha^2} \leq \alpha$, which in turn implies that $\alpha > \epsilon^{1/3}$.

Note, however, that we cannot assume that all the messages are of the same length since this will blow up the round complexity by too much. And yet, as mentioned above, we do need to assume that the error-resilient protocol is somewhat "smooth", since otherwise the communication complexity will blow up by too much. Thus, we let $\beta > 0$ be a parameter, such that in the error-resilient protocol after a message of length ℓ comes a message of length at most $\beta^{-1}\ell$. Now, an adversary corrupting $\tilde{O}(\alpha^2 \cdot \ell)$ bits of a message can cause $\beta^{-1}\ell$ bits to be obsolete. Thus, intuitively, the parties may waste $\alpha^{-2} \cdot \beta^{-1}$ bits of communication per each corruption. This, together with the fact that the adversary has an ϵ-fraction of corruption budget and the fact that the communication blows up by at most $1 + \tilde{O}(\alpha)$, implies that $\alpha^{-2} \cdot \beta^{-1} \cdot \epsilon < \alpha$, which in turn implies that

$$\alpha^3 \cdot \beta > \epsilon. \tag{1}$$

Therefore, on the one hand we would like to make β as large as possible, to improve the communication rate; on the other hand, increasing β blows up the round complexity. At first it seems that requiring this smoothing condition (i.e., that after a message of length ℓ comes a message of length at most $\beta^{-1}\ell$), will blow up the round complexity by too much. The reason is the following: Consider the real world example, where each message sent by Alice is of length ℓ, and each message sent by Bob is of length 1. Thus, to ensure that Alice is not wasting ℓ bits of communication due to a single error in Bob's message, we need to

make the protocol smooth and have Alice send her message slowly, first sending the first β^{-1} bits, then after getting a bit of approval from Bob, Alice will send the next β^{-2} bits of her message, and so on. Thus, the number of rounds it will take Alice to send her message is roughly $\log_{\frac{1}{\beta}}(\ell)$. This will cause a blowup of roughly $\log_{\frac{1}{\beta}}(\ell)$ to the round complexity, which is way too much.

To avoid this blowup, we want to make sure that after a long message does not come a message which is too short, since short messages may cause a blowup to the round complexity (if the following message is long). However, this should be done while adding at most an α fraction to the communication complexity. Thus, we also smooth the protocol in the "other direction" and require that after a message of length ℓ comes a message of length at least $\alpha\ell$. Thus, going back to our example above, where Alice is talkative (sends messages of length ℓ) and where Bob sends messages of length 1, we first convert this to another protocol where Bob sends messages of length $\alpha\ell$. This does not change the round complexity at all, and changes the communication complexity by at most an α-factor. Now, we smoothen out this protocol, by having Alice, rather than sending her ℓ bit message in "one shot", she will first send $\beta^{-1}\alpha\ell$ bits, then send the next $\beta^{-2}\alpha\ell$ bits, and so on. Note that this will cause a blowup of $\log_{\frac{1}{\beta}}(\alpha^{-1})$ in the round complexity. Since we allow a blowup of at most α' to the round complexity (without taking into account the blowup due to error, or the additive term), we take β so that $\log_{\frac{1}{\beta}}(\alpha^{-1}) \leq \alpha'$, and thus we must take β such that $\beta \leq \alpha^{1/\alpha'}$. This together with Equation (1), implies that

$$\alpha^{3+1/\alpha'} > \epsilon,$$

as in Theorem 4 above.

It remains to explain the additive term in the round blowup and the multiplicative term that depends on the round error-rate ϵ'. For the latter, clearly, if ϵ'-fraction of the rounds were completely corrupted, these rounds need to be redone, and this incurs a blowup of $1+\epsilon'$ to the round complexity. As to the former, suppose the original protocol consists of a short message followed by a very long message, to make this protocol error resilient we will have to blow up the round complexity by essentially $\log_{\frac{1}{\beta}} \mathrm{CC}$, where CC is the communication complexity of the original protocol. This is the reason we have the log additive term in the round complexity.

Finally, we explain why $\epsilon' = \alpha \cdot \mathsf{poly}(\alpha')$. We note that for the purpose of our application (Theorem 1) the exact power of α' is not important. Consider a protocol that consists of a single bit per round. In this case we can effort to add a hash check only every α^{-1} rounds. In this case, the adversary can corrupt the first message of each chunk of α^{-1} rounds, which would render the entire chunk useless. Thus, a corruption of ϵ'-fraction of the rounds, may result with a round blowup of $\epsilon'\alpha^{-1} \leq \alpha'$, which implies that indeed $\epsilon' \leq \alpha\alpha'$.

3 Smooth Protocols

Throughout this section, we refer to "rounds" in a protocol as a one way communication. Namely, the number of rounds in a protocol is equal to the number of messages that are sent in the protocol. We note that in Section 4 we diverge from this interpretation, and refer to "rounds" as a back-and-forth communication between Alice and Bob. This inconsistency allows us to simplify the notation and the presentation. Note that these two interpretations can be interchanged, while incurring a blowup of at most 2 in the round complexity.

Let Π be an arbitrary 2-party protocol. We denote by m_r the messages sent in the r^{th} round in Π. In this section we show how to convert any protocol Π into a *smooth* protocol S^{Π}. In what follows we denote by M_r the message sent in the r^{th} round in S^{Π}.

▶ **Definition 5.** *A protocol is (α, β)-smooth if for every round r the following holds:*

$$\alpha \cdot \max\{|M_{r-1}|, |M_{r-2}|, |M_{r-3}|\} \leq |M_r| \leq \frac{1}{\beta} \cdot \min\{|M_{r-1}|, |M_{r-2}|, |M_{r-3}|\} \quad (2)$$

▶ **Lemma 6.** *For any $\alpha < \frac{1}{4}$ and $\beta \leq \frac{\alpha}{8}$, the following holds: Any protocol Π can be efficiently converted into an (α, β)-smooth protocol S^{Π} such that*

1. $\mathrm{CC}(S^{\Pi}) \leq \mathrm{CC}(\Pi) \cdot (1 + 50\alpha)$.
2. $R(S^{\Pi}) \leq R(\Pi) \cdot (1 + 8\log_{2\beta} \alpha) + 4\log_{\frac{1}{2\beta}} \cdot \mathrm{CC}(\Pi) + 4$
3. *If Π is computationally efficient then so is S^{Π}.*

We defer the proof of Lemma 6 to full version.

▶ Remark 7. In Sections 4, 5, and 6, we show how to convert a smooth protocol into an error-resilient one. Similarly to previous error-resilient protocols in the literature, we will first pad the smooth protocol, and only then we convert the padded protocol into an error-resilient one. However, we will need to pad our protocol in a smooth way. This is done as follows: Suppose we want to pad our protocol with anywhere between L and $2L$ bits of 0's. Suppose that the last message in the smooth protocol is of length ℓ, then we add a message of length $\left\lfloor \frac{\ell}{\beta} \right\rfloor$, followed by a message of length $\left\lfloor \frac{\ell}{\beta^2} \right\rfloor$, and so on, until we add between βL and L bits, after which we add L bits (if we haven't added so already).

Note that such a padding results in a smooth protocol, where the communication complexity increases by at least L and at most $2L$ bits. The number of additional rounds required to do this padding is at most $\log_{\frac{1}{\beta}} L + 1$.

From now on, when we say that we convert a protocol Π to a smooth protocol, we assume that the resulting smooth protocol is padded appropriately.

4 Interactive Coding in the Ideal Hash Model

In this section, we show how to convert any protocol Π into an error resilient protocol, and analyze its properties in the **Ideal Hash Model**. This model assumes the existence of an ideal hash. In our protocol, Alice and Bob check equality of their partial transcripts, by sending to each other hashes of their partial transcripts. In this section, we consider the Ideal Hash Model, where when we analyze the communication complexity of the protocol we do not take into account the length of the hash values, and simply assume that the number of hash collisions is bounded, yet adversarially chosen. (We explain how we bound the number of collisions below). In Sections 5 and 6, we show how to remove this ideal model assumption, by implementing this ideal hash using a real hash function. In these sections, we use hash values that are short enough so the communication blowup is small, and yet we prove that with high probability the amount of hash collisions is bounded.

Moreover, we consider an adversary that either leaves a message (and corresponding hash) intact, or "fully" corrupts it. More precisely, we say that the hash is corrupted if and only if a collision occurs. In the analysis of this ideal error-resilient protocol, we say that a message is corrupted if the adversary corrupts any bit of the message (or if he corrupts the corresponding hash). We let the budget of corrupting a message be the maximum between the original message length, and the corrupted one. In particular, even if the adversary corrupted a single bit of a long message of length n (or if he corrupts only the hash corresponding to this message), we count it as n corruptions. We recall that the reason for this budgeting is that in our actual error-resilient protocol we will apply the error correcting code of Guruswami and Li [18] to each message (and hash) separately. Thus, in order to corrupt a message, the adversary will need to corrupt a constant fraction of the bits in the message. We refer the reader to Section 7 for details.

In what follows, we set

$$\alpha \leq 0.01 \ \ , \ \ \alpha' \leq 1 \ , \ \ d \geq \frac{1}{\alpha} \ \text{ and } \ \beta \leq \min\left\{\alpha^{\frac{1}{\alpha'}}, \frac{1}{5\alpha d^2}\right\}. \tag{3}$$

We assume for simplicity that α^{-1} and β^{-1} are integers. We assume without loss of generality, that the underlying protocol Π is (α, β)-smooth. This is without loss of generality since by Lemma 6, we can convert Π to an (α, β)-smooth protocol while increasing its communication complexity by a multiplicative factor of $(1 + O(\alpha))$, and increase the number of rounds by a multiplicative factor of $(1+O(\alpha'))$ and an additive factor of at most $\log \mathrm{CC}(\Pi)$, as desired.

4.1 The Protocol

We note that this (ideal) protocol is quite similar to the error-resilient protocol of Haeupler [19]. The main difference being that we need to first convert the protocol into a smooth one (whereas the protocol considered in [19] is perfectly smooth since in each round each party sends a single bit to the other party). Moreover, and more importantly, since our protocol is not perfectly smooth, when the parties backtrack, they do not erase the questionable transcript (since the messages in the questionable part may grow exponentially). Instead, the parties keep this transcript as questionable, and enter a "verification" state where they check consistency round-by-round. We note that in [19] the questionable transcript is simply erased.

In what follows, we present the (error-resilient) protocol only from Alice's perspective. Bob's perspective is symmetric. During the (error-resilient) protocol, Alice has a private variable T_A, which she believes to be a prefix of the transcript she is trying to reconstruct. T_A is initiated to $\emptyset$. We denote by m_A the message that Alice sends in the error resilient protocol.

In what follows, we define all the other notations (in addition to m_A and T_A) that are used in the protocol description:

$$S_A, R_A, \ell_A, \ell^+, \ell^-, w_A, R_A^{(1)}, R_A^{(2)},$$

where all of these variables are defined as functions of T_A and m_A.

From now on we think of each round as consisting of consecutive two messages: a message sent by Alice and a following message sent by Bob. We note that this diverges from the way we defined rounds in Section 3, where we thought of each round as containing a single message (sent by one party). The only reason for this inconsistency is that it is more convenient in terms of notations. It is important to note that this is only a notational convenience and does not affect our final result in any way.

For each variable used in our protocol

$$v_A \in \{T_A, m_A, S_A, R_A, \ell_A, \ell^+, \ell^-, w_A, R_A^{(1)}, R_A^{(2)}\},$$

we denote by $v_{A,r}$ the value of v_A that Alice uses when sending her round r message, and we occasionally omit r when it is clear from the context.

- During the protocol Alice has a state

  $$S_A \in \{\text{Simulation}, \text{Verification}\} \cup \mathbb{N}.$$

  Loosely speaking, Alice is in a Simulation state when she believes that the transcript T_A that she is holding is indeed a prefix of the correct transcript.

If $S_A \in \mathbb{N}$ then we say that Alice is in a Correction state. If Alice is in Correction state, then S_A is the first round (in the error-resilient protocol) that Alice has entered this state. Alice enters a Correction state when she thinks her beliefs are wrong (for example, when the hashes indicate that T_A and T_B are inconsistent). During this state, Alice tries to go back to an earlier round in the transcript (corresponding to the original protocol) which she believes to be correct. We denote this round by R_A. Alice will continue the simulation from $T_A[R_A]$, which denotes the truncated transcript of T_A to round R_A. As mentioned above, as opposed to the protocol of Haeupler [19], in our protocol, she does not delete the suffix of T_A, and rather she keeps this suffix as questionable. The reason she does not erase this questionable suffix, is that it may be the correct suffix (and the only reason it is questioned is due to an error), and in this case it may be too expensive to delete and reconstruct, since in our case the messages in the questionable suffix may grow at an exponential rate.

After a Correction state, Alice enters either another Correction state or a Verification state, where she decides whether to completely delete, partially delete, or keep, the questionable suffix. After the Verification state (assuming there were no errors), Alice enters Simulation state again.

We define $r - S_A$ to be zero, when $S_A \in \{\text{Simulation}, \text{Verification}\}$.

- As mentioned above, R_A denotes the round in T_A that Alice simulates. If $S_A = \text{Simulation}$ then R_A is equal to the number of rounds in T_A.
- Let $m_{A,r}$ be the message that Alice sends in round r (of the error resilient protocol), and let $\ell_{A,r}$ denote its size. Let $m_{B,r}$ be the message that Alice received from Bob in round r, and let $\ell_{B,r}$ denote its size. We define $\ell_{\max,r} = \max\{\ell_{A,r}, \ell_{B,r}\}$. Note that if Bob's message was corrupted then $\ell_{B,r}$ may be arbitrarily large. However, our (error-resilient) protocol has the property that if Bob's message was not corrupted then $\ell_{B,r} \leq \frac{\ell_{A,r}}{\beta}$.
  We define

$$\ell_r^+ = \min\left\{\frac{\ell_{A,r}}{\beta}, 2\ell_{\max,r}\right\}$$

  and

$$\ell_r^- = \min\left\{\frac{\ell_{A,r}}{\beta}, \max\left\{\beta^{-1}, \alpha\ell_{\max,r}\right\}\right\}.$$

- Let $w_{A,r}$ be $2^{\lfloor \log(r-S_A)\rfloor}$ if $S_{A,r} \in \mathbb{N}$, and let $w_{A,r}$ be 0 otherwise. In other words, $w_{A,r}$ is the number of rounds that the party has been in Correction state, rounded to the closest power of two.
- If $w_A = 0$ then let $R_A^{(1)} = R_A$. Otherwise, let $R_A^{(1)} < R_A$ be maximal that divided w_A.
- $R_A^{(2)} \triangleq R_A^{(1)} - w_A$.
- In the protocol, at each round r, Alice sends hashes to Bob if and only if $r = 0 \pmod d$ or $\ell_{A,r} \geq d$, in which case she sends five hashes, one hash corresponding to each of the following strings:

$$\left(T_A[R_A], T_A[R_A+1], T_A[R_A^{(1)}], T_A[R_A^{(2)}], S_A\right).$$

  We note that if R_A is equal to the number of rounds in T_A, then Alice will not know the partial transcript $T_A[R_A+1]$. In this case we define $T_A[R_A+1] = T_A[R]$.

Alice in round r. Upon receiving a message from Bob, parse the message as

$$(m_{B,r-1}, H(T_{B,r-1}[R_{B,r-1}]), H(T_{B,r-1}[R_{B,r-1}+1]),$$
$$H(T_{B,r-1}[R^{(1)}_{B,r-1}]), H(T_{B,r-1}[R^{(2)}_{B,r-1}]), H(S_{B,r-1})).$$

We assume that in this ideal model, parsing is easy. When we implement this ideal hash function in Section 5, we will make sure that indeed Alice will be able to parse correctly (assuming the message was not corrupted). Denote the size of $m_{B,r-1}$ by $\ell_{B,r-1}$.

1. If $S_{A,r-1} =$ Simulation then do the following:

   a. If a hash was sent by Bob (i.e., if $\ell_{B,r-1} \geq d$ or d divides $r-1$) then check that

   $$H(S_{B,r-1}) = \text{Simulation} \quad \text{and} \quad H(T_{A,r-1}[R_{A,r-1}]) = H(T_{B,r-1}[R_{B,r-1}]).$$

   b. Check that the (partial) transcript $(T_{A,r-1}[R_{A,r-1}], m_{A,r-1}, m_{B,r-1})$ satisfies the (α, β)-smoothness condition.
   c. If one of these conditions does not hold then let
      - $m_{A,r} = 0^{\ell^-_{r-1}}$
      - $S_{A,r} = r$
      - $(T_{A,r}, R_{A,r}) = (T_{A,r-1}, R_{A,r-1})$.
   d. Else, set
      - $T_{A,r} = (T_{A,r-1}, m_{A,r-1}, m_{B,r-1})$
      - $R_{A,r} = R_{A,r-1} + 1$
      - $m_{A,r} = \Pi(T_{A,r})$
      - $S_{A,r} = S_{A,r-1} =$ Simulation.

2. If $S_{A,r-1} =$ Verification then check if all the following conditions hold:

   a. $|m_{B,r-1}| \geq \beta^{-1}$.
   b. $H(S_{A,r-1}) = H(S_{B,r-1})$
   c. $H(T_{A,r-1}[R_{A,r-1}]) = H(T_{B,r-1}[R_{B,r-1}])$.

   If one of these conditions does not hold then let
   - $m_{A,r} = 0^{\ell^-_{r-1}}$
   - $S_{A,r} = r$
   - $(T_{A,r}, R_{A,r}) = (T_{A,r-1}, R_{A,r-1})$.

   Else, do the following:

   a. If number of rounds in $T_{A,r-1}$ is greater than $R_{A,r-1}+1$, and

   $$H(T_{A,r-1}[R_{A,r-1}+1]) = H(T_{B,r-1}[R_{B,r-1}+1]),$$

   then let
   - $m_{A,r} = 0^{\ell^-_{r-1}}$
   - $R_{A,r} = R_{A,r-1} + 1$
   - $(S_{A,r}, T_{A,r}) = (S_{A,r-1}, T_{A,r-1})$.

   b. Else, if $m_{A,r-1} = m_{B,r-1} = 1^\ell$ for some $\ell > |T_{A,r-1}| - |T_{A,r-1}[R_{A,r-1}]|$, then set
   - $T_{A,r} = T_{A,r-1}[R_{A,r-1}]$
   - $R_{A,r}$ be the number of rounds in $T_{A,r}$
   - $m_{A,r} = \Pi(T_{A,r})$
   - $S_{A,r} =$ Simulation.

c. Else, if $\ell^+_{r-1} > |T_{A,r-1}| - |T_{A,r-1}[R_{A,r-1}]|$ then let
- $m_{A,r} = 1^{\ell^+_{r-1}}$
- $(T_{A,r}, R_{A,r}, S_{A,r}) = (T_{A,r-1}, R_{A,r-1}, S_{A,r-1})$.

d. Else, let
- $m_A = 0^{\ell^+_{r-1}}$
- $(T_{A,r}, R_{A,r}, S_{A,r}) = (T_{A,r-1}, R_{A,r-1}, S_{A,r-1})$.

3. Else, do the following:

a. Compute the values v^0_r, v^1_r, v^2_r as follows:
If $H(S_{A,r-1}) \neq H(S_{B,r-1})$ then set $v^0_r \leftarrow v^0_{r-1} + 1$.
Else, if $H(T_{A,r-1}[R^{(1)}_{A,r-1}]) \in \left\{H(T_{B,r-1}[R^{(1)}_{B,r-1}]), H(T_{B,r-1}[R^{(2)}_{B,r-1}])\right\}$ then set $v^1_r \leftarrow v^1_{r-1} + 1$.
Else, if $H(T_{A,r-1}[R^{(2)}_{A,r-1}]) \in \left\{H(T_{B,r-1}[R^{(1)}_{B,r-1}]), H(T_{B,r-1}[R^{(2)}_{B,r-1}])\right\}$ then set $v^2_r \leftarrow v^2_{r-1} + 1$.

b. If $r - S_{A,r-1}$ is not a power of 2,[9] then set
- $m_{A,r} = 0^{\ell^-_{r-1}}$
- $(T_{A,r}, R_{A,r}, S_{A,r}) = (T_{A,r-1}, R_{A,r-1}, S_{A,r-1})$.

c. Else, if $v^0_{r-1} > \frac{1}{2}(r - S_{A,r-1})$ then let
- $S_{A,r} = r$
- $m_{A,r} = 0^{\ell^-_{r-1}}$
- Set $v^0_r = v^1_r = v^2_r = 0$.

d. Else, if $v^1_r > \frac{1}{4}(r - S_{A,r-1})$ then let
- $S_{A,r} =$ Verification
- $R_{A,r} = R^{(1)}_{A,r-1}$
- $m_{A,r} = 0^{\ell^-_{r-1}}$
- Set $v^0_r = v^1_r = v^2_r = 0$.

e. Else, if $v^2_r > \frac{1}{4}(r - S_{A,r-1})$ then let
- $S_{A,r} =$ Verification
- $R_{A,r} = R^{(2)}_{A,r-1}$
- $m_{A,r} = 0^{\ell^-_{r-1}}$
- Set $v^0_r = v^1_r = v^2_r = 0$.

f. Else, set $v^1_r = v^2_r = 0$, and let
- $m_{A,r} = 0^{\ell^-_{r-1}}$
- $(v^0_r, R_{A,r}, S_{A,r}, T_{A,r}) = (v^0_{r-1}, R_{A,r-1}, S_{A,r-1}, T_{A,r-1})$.

Send $m_{A,r}$, and if $r = 0 (\text{mod } d)$ or $|m_{A,r}| \geq d$ then append to $m_{A,r}$ also

$$\left(H(T_{A,r}[R_{A,r}]), H(T_{A,r}[R_{A,r}+1]), H(T_{A,r}[R^{(1)}_{A,r}]), H(T_{A,r}[R^{(2)}_{A,r}]), H(S_{A,r})\right).$$

Remark. Bob behaves identically to Alice, except in Steps 1 and 2b, when Bob computes his next message corresponding to the underlying protocol Π, he computes it by $m_{B,r} = \Pi(T_{B,r}, m_{A,r})$, whereas recall that Alice computed it by $m_{A,r} = \Pi(T_{A,r})$.

[9] Recall that we define $r - S_A = 0$ if $S_A \in \{\text{Simulation}, \text{Verification}\}$, and we consider 0 to be power of 2.

4.2 Analysis

Terminology. In what follows, we introduce terminology that we use in the analysis.

We allow the adversary to create collisions in the ideal hash function, in which case we say that the hash was corrupted. We say that a message is corrupted if the adversary corrupts any bit of the message, or corrupts the associated hash. We define the budget of corrupting a message m to be the maximum between the length of m and the length of the corrupted version of m. Thus, even if the adversary corrupts a few bits of a long message of length n (or corrupts the associated hash), then we count it as n corruptions. On the other hand, if the adversary corrupted a single bit message by converting it into a long n-bit message, then we count it as n corruptions.

We analyze the correctness of the (error-resilient) protocol assuming a bound on these message corruptions.

▶ **Definition 8.** *We say that the corrupted messages have* volume e *if the sum of lengths of corrupted messages (where each such length is the maximum between the length of the original message and the length of the corrupted version of it) is e.*

Using this terminology we prove the following theorem.

▶ **Theorem 9.** *Let $\Pi = (A, B)$ be any (α, β)-smooth protocol, and let $\Pi' = (S^A, S^B)$ be the simulated protocol defined above. Let $\mathcal{A}$ be any adversary in Π', who corrupts at most e' messages of total volume of at most e. Then, the protocol Π', executed with the adversary $\mathcal{A}$, denoted by $\Pi'_{\mathcal{A}}$, satisfies the following.*

1. $CC(\Pi'_{\mathcal{A}}) \geq t_{\min}$*, where $t_{\min}$ is a lower bound of the communication of any instance of* Π.
2. $CC(\Pi'_{\mathcal{A}}) \leq CC(A, B) + 18\beta^{-1}e + 20d\beta^{-1}e'$.
3. $R(\Pi'_{\mathcal{A}}) \leq R(A, B) + 906d \log \frac{1}{\beta} e'$.
4. *The parties outputs transcripts of size at most* $\mathrm{CC}(\Pi'_{\mathcal{A}})$ *that agree with* Π *on the first*

$$CC(\Pi'_{\mathcal{A}}) - 18\beta^{-1}e - 20d\beta^{-1}e' \ ,$$

   many bits.
5. *S is a polynomial time oracle machine.*

▶ Remark 10. We will apply Theorem 9 with an adversary $\mathcal{A}$ that corrupts at most $e' = O(\min\{\epsilon' \cdot R(\Pi'_{\mathcal{A}}), \frac{\epsilon}{d} CC(\Pi'_{\mathcal{A}})\})$ messages of total volume at most $e = O(\epsilon \cdot CC(\Pi'_{\mathcal{A}}))$, where $\epsilon \leq O(\alpha \cdot \beta)$ and $\epsilon' \leq \frac{\alpha'}{d \cdot \log \beta^{-1}}$. Thus,

$$\begin{aligned}
&CC(\Pi'_{\mathcal{A}}) \leq \\
&CC(A, B) + 18\beta^{-1}e + 20d\beta^{-1}e' \leq \\
&CC(A, B) + 0(\beta^{-1}\epsilon \cdot CC(\Pi'_{\mathcal{A}})) + O\left(d\beta^{-1}\frac{\epsilon}{d} CC(\Pi'_{\mathcal{A}})\right) \leq \\
&CC(A, B)(1 + O(\alpha)),
\end{aligned}$$

and

$$\begin{aligned}
&R(\Pi'_{\mathcal{A}}) \leq \\
&R(A, B) + 906d \log \frac{1}{\beta} e' \leq \\
&R(A, B) + O(d \log \frac{1}{\beta} \epsilon' \cdot R(\Pi'_{\mathcal{A}})) \leq \\
&R(A, B)(1 + O(\alpha')),
\end{aligned}$$

as desired.

Moreover, we will apply this theorem with a protocol Π which is padded by $18\beta^{-1}e + 20d\beta^{-1}e' = O(\alpha \cdot CC(\Pi'_{\mathcal{A}}))$ zeros. Thus, Item 4 from Theorem 9 implies correctness.

For example, one can set $\alpha = O(\min\{\sqrt{\epsilon}, (\epsilon')^{1/3}\})$, and set $\beta = O(\alpha)$, $d = \frac{1}{\alpha}$, $\alpha' = 1$, to obtain an error resilient protocol in the ideal hash model with constant blowup in round complexity and $1 + O(\alpha)$ blowup in communication complexity.

We defer the proof of Theorem 9 to full version.

5 Hash Implementation with Shared Randomness

Recall that in Section 4, we presented an interactive coding scheme with the desired guarantees, in the ideal hash model, where we assume that the number of hash collisions is bounded, and where the budget for making a collision is proportional to the message length (where the message length is the maximum between the length of the message that was sent and the corrupted version of it). We denote this ideal protocol by Π.

In this section, we show how to implement the ideal hash with a real hash function. Loosely speaking, given a hash function h, we convert the protocol Π to the protocol Π^h which is identical to Π, where the ideal hash function is replaced by h. In order to maintain the desired efficiency and error-resilience guarantees, we need to ensure that, on the one hand, these hash values are not too long; and on the other hand there are not too many hash collisions (i.e., that these hashes form a good equality test). To ensure the latter condition holds, it is easy to see that we cannot use a single (deterministic) hash function. Instead we use a *family* of *randomized* hash functions.

We construct a function family $\mathcal{H} = \{h_x\}$, where each hash function h_x is associated with a (possibly long) seed x. In this section, we consider the *shared randomness model*, where the parties are allowed to share a (possibly long) random string. In Section 6 we show how to eliminate the need for shared randomness.

In this section we assume that the shared randomness is as long as we need. In particular, we use a different hash function (i.e, a different seed) for each equality query. Since the length of the protocol is adaptive and not a priori bounded[10], the length of the common random string is also not a priori bounded. We assume that there is a separate segment of the common random string for each round r, and each such segment contains five hash seeds, since in Π, in rounds that a party sends an ideal hash, the party sends five ideal hashes.

We emphasize that the shared randomness (and in particular the seeds) are known to the adversary. Therefore the adversary, given a seed x can try to skew the protocol and cause the parties to send many messages whose hashes collide. To get around this, we construct a hash family, where each h_x is a *randomized* hash function. When a party sends a hash of a value V, the party will choose randomness S and will send $(S, h_x(V, S))$. On the one hand, the randomness S needs to be short, since otherwise this will blowup the communication complexity by too much. On the other hand, the adversary cannot predict S, and thus will not be able to skew the messages of the parties towards ones which the hashes collide. We note that a similar idea of using a randomized hash function was used by Haeupler [19], for the sake of improving the rate of his interactive coding scheme.

Before presenting our randomized hash family, we start with some preliminaries.

[10] In Section 6, we convert any such protocol in the *unbounded* shared randomness model into one that uses only private randomness.

5.1 Preliminaries

Chernoff bounds.

▶ **Lemma 11.** *For any $N \in \mathbb{N}$, and any N independent Bernoulli random variables $X_1, \ldots, X_N$, each with mean $\leq \gamma$, it holds that*

$$\Pr[\sum_{i=1}^{N} X_i > 2\gamma N] \leq e^{-\frac{1}{3}\gamma N}.$$

▶ **Definition 12.** *A distribution D over $\mathbb{F}_2^n$ is* δ-bias *if for any $v \in \mathbb{F}_2^n \setminus \{0^n\}$, we have that*

$$\left| \Pr_{x \sim D} \left[\sum_{i=1}^{n} v_i x_i = 0 \right] - \frac{1}{2} \right| \leq \delta \ .$$

▶ **Lemma 13** ([25])**.** *There exists an absolute constant $C \in \mathbb{N}$ and an efficiently computable function $G : \{0,1\}^* \to \{0,1\}^*$ such that for any size k and a uniformly random string $S \in \{0,1\}^{Ck}$, we have that $G(S) \in \{0,1\}^{2^k}$ is a 2^{-k}-biased distribution of length 2^k.*

▶ **Lemma 14** (6.3 from [19])**.** *There exists a hash family $\mathcal{F} = \{\mathcal{F}_L\}_{L \in \mathbb{N}}$, such that for every $L \in \mathbb{N}$ it holds that $\mathcal{F}_L = \{f_x\}_{x \in \{0,1\}^{2L}}$, and for every $x \in \{0,1\}^{2L}$, $f_x : \{0,1\}^{\leq L} \to \{0,1\}$. Moreover, for any $k \in \mathbb{N}$ and for any vectors $V_1^A, ..., V_k^A, V_1^B, ..., V_k^B \in \{0,1\}^{\leq L}$ the following holds:*

1. *For uniform $x = (x_1, \ldots, x_k) \in (\{0,1\}^{2L})^k$ it holds that for each $i \in [k]$, the probability that $f_{x_i}(V_i^A) = f_{x_i}(V_i^B)$ is $\frac{1}{2}$ whenever $V_i^A \neq V_i^B$, and 1 whenever $V_i^A = V_i^B$. Moreover, for each $i \in [k]$ these probabilities are independent.*
2. *For δ-biased distribution $x = (x_1, \ldots, x_k) \in (\{0,1\}^{2L})^k$, it holds that the distribution*

$$\left(\mathbf{1}_{f_{x_1}(V_1^A) = f_{x_1}(V_1^B)}, \ldots, \mathbf{1}_{f_{x_k}(V_k^A) = f_{x_k}(V_k^B)} \right)$$

is δ-close to the case where x is uniform.

5.2 Our Hash Function

We are now ready to construct our family of randomized hash functions. We first define the randomized hash family $\mathcal{H}' = \{h'_x\}$, which uses the hash family $\mathcal{F}$ from Lemma 14. Recall that the hash values of $\mathcal{H}'$ should not be too long, since this will result in a large blowup in communication complexity.

In our construction, as opposed to previous constructions [2, 3, 19], the length of each hash value depends not only on the length of the message it is appended to, but it also depends on the length of the *entire* communication up until the point that the hash was sent. We note that if we were only concerned with the communication blowup and were not concerned with the round blowup, then we could have the length of the hash value depend only on the length of the message it is sent with (in similar spirit to prior work). However, as we argue below, in order to ensure a constant blowup in round complexity, we must allow the length of the hash value to also depend on the length of the entire history. This is illustrated in the following example: Suppose that a short message is sent, and prior to this short message were a few very long messages (in a way that satisfies the smoothness criterion). By corrupting a few long messages, the adversary can cause a hash collision in many short messages (with hash), which will result with a large blowup to the round complexity.

Hence, we allow the length of the hash value, not only to depend on the length of the message it is appended to, but also to depend on the length of the communication history. Note that the parties do not necessarily agree on the history length even if no errors occur in the current round, since the adversary may insert and delete bits throughout the protocol, in which case they will fail to parse the message and hash pair correctly.

Thus, we define the hash family $\mathcal{H} = \{h_x\}$, where the output of h_x includes the output of h'_x, the randomness used by h'_x (which is needed in order check for equality), and also the length of the hash value. Namely, we define

$$h_x(V) = (S, h'_x(V;S), 1 \cdot 0^w),$$

where S is the (private) randomness used by h'_x, and w is the length of $H'_x(V;S)$.

We next define h'_x. As we mentioned, the length of the hash values (denoted by w) may differ from one round to the next, as they depend on the communication complexity so far, and on the length of the current message sent. We will specify how w is defined below. But we first, define h'_x assuming w is known.

The seed x is random in $\left(\{0,1\}^{2L}\right)^L$, where we assume that L is greater than the input V (which is bounded by the communication complexity of the protocol up until the point where the hash is sent). For any $y = (y_1, \ldots, y_L) \in \left(\{0,1\}^{2L}\right)^L$ and for any $k \leq L$, let

$$f^k_y(V) = (f_{y_1}(V), ..., f_{y_k}(V)).$$

where $\mathcal{F} = \{f_y\}$ is the hash family from Lemma 14. The randomness for h'_x is denoted by S and is of size $2C \cdot w$. Let h'_x be the randomized hash function, that takes as input a variable V, randomness S, and outputs

$$h'_x(V;S) = f^w_{G(S)}(Z),$$

where

$$Z = \begin{cases} (f_x^{2^w}(V), 0) & \text{if } |V| \geq 2^w \\ (V, 1) & \text{if } |V| < 2^w \end{cases}$$

In what follows we show how the length w of the hash values are chosen. To this end we need to define the following variables with respect to a certain round r.

- Let Q^A be the set of all rounds r in which Alice send a hash to Bob.
  Note that these are exactly the set of rounds r such that r divides d or Alice send a message of length $\geq d$.
  We define Q^B analogously.
- Let a^A_r be the number of messages that Alice sent with a hash until (and including) round r. Namely,

  $$a^A_r = |\{r' \leq r : r' \in Q^A\}|.$$

  We define a^B_r analogously.
- Let t^A_r be all the communication received by Alice until (and including) round r. We define t^B_r analogously.
- For every r, if Alice sends the message in round r then define

  $$u_r = \max_{r' \in Q^A \cap [r-1]} \log \frac{t^A_r - t^A_{r'}}{a^A_r - a^A_{r'}}.$$

  The definition is analogous in the case that Bob sends the message in round r.

- For every round r, let ℓ_r denote the length of the message to be sent in round r, and let

$$w_r = \lceil \alpha \ell_r \rceil + \lceil u_r \rceil + 9 \left\lceil \log \frac{1}{\gamma} \right\rceil + 6,$$

where $\alpha, \gamma > 0$ are parameters of the scheme, where $\gamma < \alpha$ (it will be instructive to think of $\gamma = \epsilon$, where ϵ is the corruption budget of the adversary, and of α as the communication blowup in the error-resilient protocol).

5.3 Analysis

We denote by E the set of all messages $m_{A,r}$ or $m_{B,r}$ that were not corrupted but had a hash associated with them that formed a hash collision. Recall that for any set of messages T, we denote by $|T|$ the volume of T (i.e., the number of bits in T), and we denote by $|T|'$ the number of messages in T.

▶ **Lemma 15.** *Fix $\epsilon < 0.0005$. The protocol $\Pi^{\mathcal{H}}$ defined in Section 5.2 satisfies the following: If $\Pi^{\mathcal{H}}$ consists of $\leq t$ bits and $\leq r$ rounds, then for any adversary that corrupts messages with total volume at most ϵt, we get that*

1. *With probability $\geq 1 - 10 \cdot e^{-\frac{\alpha\gamma}{3 \log \frac{1}{\gamma}} t}$ (over the common and private randomness),*

$$|E| \leq 20\gamma t.$$

2. *With probability $\geq 1 - 10e^{-\frac{2}{3}\gamma r}$ (over the common and private randomness),*

$$|E|' \leq 70\gamma r.$$

5.4 Communication Bound

In this section we will bound the blowup of the communication of $\Pi^{\mathcal{H}}$, defined in Section 5.2. To this end, fix any adversary $\mathcal{A}$ for the protocol $\Pi^{\mathcal{H}}$, that corrupts at most e' messages of total volume at most e. We define a corresponding adversary $\mathcal{D}$ for the protocol Π, that corrupts at most e' message of total volume at most e, as follows:

The adversary $\mathcal{D}$ sends the exact same messages as $\mathcal{A}$ does, excluding the hash values. Recall that for each message in $\Pi^{\mathcal{H}}_{\mathcal{A}}$, the part that belongs to the hash value is well-defined by the suffix of the message $1 \cdot 0^w$, and hence the adversary $\mathcal{D}$ is well defined.

▶ **Lemma 16.**

$$\mathrm{CC}(\Pi^{\mathcal{H}}_{\mathcal{A}}) \leq (1 + 50C\alpha)\,\mathrm{CC}(\Pi_{\mathcal{D}}) + e + 600C \log \frac{1}{\gamma} \cdot k$$

where C is the universal constant from Lemma 13, and k is the number of rounds with hash in $\Pi_{\mathcal{D}}$.

The proof of this lemma is deferred to full version.

6 Hash Implementation with Private Randomness

In this section we show how to implement the ideal hashes in protocol Π, defined in Section 4, without resorting to shared randomness, but rather using only private randomness. To this end, we will slightly modify the protocol Π, into a new protocol Π'.

High-level overview of Π'. Lets first recall the approach used in previous works [2, 19]. In these works, the long shared randomness is replaced with δ-biased randomness, where $\delta = 2^{-\alpha t}$ and t is a bound on the communication complexity. Such δ-biased randomness can be generated using only $O(\alpha t)$ random bits. Hence, in previous works, these $O(\alpha t)$ bits of randomness are sent in advance (using an error correcting code). If we indeed had a bound t on the communication complexity, then this idea would work, as explained below.

Recall that the randomness is used for equality testing. From Lemma 14, we know that for any oblivious adversary (i.e., one that is independent of the randomness), the fraction of collisions in the case where the seed is random is δ-close to the fraction of collisions in the case where the seed is δ-biased. Denoting by N the number of possible oblivious adversaries, and by taking a union bound over all possible oblivious adversaries, we conclude that the probability that there exists an oblivious adversary that causes "too many" hash collisions in the case where the seed is δ-based is bounded by the same probability where the seed is truly random plus an additive term of δN. We note that

$$N \leq 2^{H(\epsilon)t} \cdot 4^{\epsilon t} = 2^{O(\epsilon \log \frac{1}{\epsilon} t)},$$

and thus

$$\delta N = 2^{-\alpha t} \cdot 2^{O(\epsilon \log \frac{1}{\epsilon} t)} = 2^{-\Omega(\alpha t)}.$$

Therefore, the probability that there exists an oblivious adversary that causes "too many" hash collisions is at most $2^{-\Omega(\alpha t)}$. Note that we can view any (non-oblivious) adversary as one that chooses an oblivious adversary as a function of the public randomness, and runs this oblivious adversary. Therefore, we conclude that for any (non-oblivious) adversary the probability that there are "too many" hash collisions is bounded by $2^{-\Omega(\alpha t)}$.

However, in our setting, we do not have an a priori bound on the communication complexity. In particular, if we replace the CRS with δ-biased randomness, where $\delta = 2^{-\alpha t}$ for some t, and if the adversary has a corruption budget of more than $O(\alpha t)$ bits (i.e., the communication complexity is larger than $\frac{\alpha t}{\epsilon}$), then our protocol is no longer safe. We overcome this problem by sending more randomness as the communication complexity increases.

More specifically, the parties start by assuming that the communication complexity is some small $t_{\min}$, where $t_{\min}$ is a lower bound on the communication complexity. So, the protocol starts when one of the paries, say Alice, chooses a random string $s \in \{0,1\}^{\alpha t_{\min}}$, and sends it to Bob.[11]

Once $t_1 \geq \frac{\alpha t_{\min}}{\epsilon}$ bits are sent in the protocol, the safety of this randomness could be compromised, since the adversary has enough budget to compromise $\alpha t_{\min}$ bits. Hence, each party, before sending its message, will check whether sending this message will cause the communication complexity to exceed $\frac{\alpha t_{\min}}{\epsilon}$. If so, then instead of sending the message, the party will send new randomness. This time, the party will choose at random s_2 of size $\alpha t_1 - |s_1|$ and send (s_1, s_2). If this randomness is inconsistent with the first randomness sent (s_1) then the party receiving the randomness aborts.

Once the communication complexity is $t_2 \geq \frac{\alpha t_1}{\epsilon}$, again the safety of the previous randomness could have been compromised, and hence as above, if a party is about to send a message that will cause the communication complexity to exceed $\frac{\alpha t_1}{\epsilon}$, then instead of sending the message, the party will choose at random s_3 such that $|s_1| + |s_2| + |s_3| = \alpha t_2$,

[11] As before, we ignore the error-correcting code, since we consider only message adversaries, that corrupt messages as opposed to bits, and the budget for corrupting a message is the length of the message (or the length of the corrupted message, whichever is longer).

and will send (s_1, s_2, s_3), etc. If at any point the randomness received is inconsistent with the previous random string then the party aborts. We refer to these special messages that transmit randomness by *system messages*.

There is a slight problem with this idea: How does Bob know which message sent by Alice corresponds to a message in the initial protocol Π, and which is a system message? We fix this problem by appending 1's to system messages, and appending 0's to messages corresponding to Π. However, recall, that we do not want to blowup the communication complexity. Hence we only append these bits to long messages. This is enough, since system messages are always long.

Note that according to our protocol the parties first receive randomness s_1, then they receive new randomness (s_1, s_2), and so on. We ensure that if at any point, a system message was decoded incorrectly, then eventually the paries will abort, and "catch" the adversary with injecting too many errors. This guarantee simplifies the analysis: Either at some point a system message was decoded incorrectly, in which case the adversary is "caught" with injecting too many errors, or all the parties always agree on the randomness, in which case correctness follows from the correctness of the underlying protocol in the shared randomness model.

To ensure that indeed the parties will always notice when a system message was corrupted, we add to the system message the rounds $r_1, \ldots, r_k$ in which system messages were sent. This is done to circumvent the case where the message (s_1, s_2) was corrupted and converted into a protocol message, and a few rounds later a protocol message was corrupted and converted into the same system message (s_1, s_2). If we do not include the round number then the parties may never notice that there was a point in the protocol where they did not agree on the shared randomness. In order to avoid dealing with such cases, we simply include the round numbers of the system messages.

Finally, we notice that even though we ensure that the parties always agree on the shared randomness (assuming the adversary does not inject too many errors), there is still a subtle issue. Note that the first random string s_1 is δ-biased for $\delta = 2^{-\alpha t_{\min}}$. As we saw in previous work, this suffices if the number of oblivious adversaries, restricted to the first $t_{\min}$-bits, is bounded by $2^{O(\alpha t_{\min})}$. However, in our setting, since the total communication may be significantly larger than $t_{\min}$, the number of such oblivious adversaries can be as large as $2^{t_{\min}}$, in which case the number of rounds with hash collisions can be large. To overcome this problem, we ensure that in the first $t_{\min}$ bits of communication, the adversary cannot inject too many errors (without being "caught"). This is done by re-sending the first $t_{\min}$ bits after $t_{\min}/\alpha$ bits of the protocol were transmitted, and the parties abort if these $t_{\min}$ bits are ϵ/α-far from the first $t_{\min}$ bits of the transcript. More precisely, to each system message sent after t bits of the protocol were communicated, we append the first αt bits of the transcript.

In what follows we present our protocol Π'. For the sake of simplicity, after each system message is sent, the party receiving a system message replies with an "echo" message, by simply repeating the system message. The purpose of this "echo" message is simply to allow the other party to send his protocol message (which he didn't have the budget to send in the previous round).

The protocol Π'. Let $b > 2$, and let $\alpha \leq \frac{1}{3200C}$, where C is the constant defined in Lemma 13. Fix any $d \in \mathbb{N}$ and $\gamma > 0$ such that

$$\gamma \leq \min\left\{\frac{1}{d}, 2^{-b}\right\} \quad \text{and} \quad d \geq \frac{\log \frac{1}{\gamma}}{\alpha}. \tag{4}$$

For convenience the reader can think of $b = 2$ and $\gamma = \frac{1}{d}$.

Let Π be the protocol, in the ideal hash model, defined in Section 4, instantiated with α and d as above, and with any $\alpha' > 0$. Let $t_{\min}$ be a lower bound on the communication complexity of Π, where

$$t_{\min} \geq \max\{\alpha^{-2}, 250C\alpha^{-1}\log d\}, \tag{5}$$

and let

$$W = \alpha t_{\min}.$$

Let $\mathcal{H}$ be the hash family defined in Section 5. The protocol Π' makes oracle access to the protocol $\Pi^{\mathcal{H}}$ (defined in Section 5).

In protocol Π', each party maintains a transcript T initialized to $\emptyset$, an integer k initialized to 0, k strings $s_1, ..., s_k$, k partial transcripts $P_1, \ldots, P_k$, and k rounds $r_1, ..., r_k$ that will be determined during the protocol. Intuitively, T is the transcript corresponding to Protocol $\Pi^{\mathcal{H}}$, $s_1, \ldots, s_k$ are k seeds that are used to generate the hash function implementing the ideal hash, and $r_1, \ldots, r_k$ correspond to rounds in $\Pi^{\mathcal{H}}$ where the common randomness changes. Similarly to Π (and $\Pi^{\mathcal{H}}$), in Π' we interpret the (partial) transcripts as strings.[12] .

In Π', if a party aborts, it always waits until at least $t_{\min}$ bits are sent before aborting the protocol, so as to fulfill the requirement that the communication complexity of Π' is at least $t_{\min}$.

In the first round of Π' Alice does the following:

1. Choose $s_1 \in_R \{0,1\}^W$, and let $k = 1$, $r_1 = 0$, and $P_1 = \emptyset$.
2. Send $(s_1, P_1, r_1, 1)$.

We next describe the protocol from Alice's point of view, given her private state

$$(T, k, s_1, \ldots, s_k, r_1, \ldots, r_k, P_1, \ldots, P_k).$$

Bob's view is symmetric (by switching between A and B). Upon receiving a message m_B, Alice does the following

1. If in the previous round Alice computed her message in step 4(e)ii of the protocol (or if the previous round was the first round of the protocol) then check that m_B is an echo of (i.e., equal to) the message sent by Alice in the previous round. If not then halt, and otherwise goto Step 4c.
2. Otherwise, denote $\ell = |m_B|$.
3. If $\ell \geq b^k W$ and the least significant bit of m_B is 1, then do the following:

   a. If there exists $s, P, r \in \{0,1\}^{b^k W}$, where r is a binary representation of $|T|'$, such that

   $$(s_1, ..., s_k, s, P_1, ..., P_k, P, r_1, ..., r_k, r, 1) = m_B,$$

   and such that P can be obtained from a prefix of T by corrupting messages of volume at most $\frac{|P|}{bd\log d}$, then define $s_{k+1} = s$, define $r_{k+1} = r$, $P_{k+1} = P$, update $k \leftarrow k+1$, and send (an echo message) m_B.

   b. Else, abort the protocol.

[12] This is done by standard encoding, where after each bit of the transcript we add a bit that represents whether the message ended or not. Thus, a transcript of length ℓ can be described by a string of length 2ℓ.

4. Else, do the following:
   a. If $\ell \geq b^k W$ then let m'_B be the message m_B when the least significant bit of m_B is truncated. Otherwise, let $m'_B = m_B$.
   b. Update $T \leftarrow T \cup \{m'_B\}$
   c. Define $m_A = \Pi^{\mathcal{H}}(T)$, using $x = x(s_1, \ldots, s_k, r_1, \ldots, r_k)$ as the shared randomness, where the exact function x is defined later. If there is no m_A to send then abort.
   d. If $|T \cup m_A| < \frac{b^k W}{400C\alpha}$ then do the following:
      i. Update $T \leftarrow T \cup \{m_A\}$.
      ii. Let $\ell = |m_A|$.
      iii. If $\ell < b^k W$ then send m_A, and otherwise send $(m_A, 0)$.
   e. Else,
      i. Let $s_{k+1}, P_{k+1}, r_{k+1} \in \{0,1\}^{b^k W}$ such that s_{k+1} is a uniformly chosen random string, P_{k+1} consists of the first $b^k W$ bits of T (where T is viewed as string) and r_{k+1} is a binary representation of $|T|'$.[13]
      ii. Send

$$(s_1, \ldots, s_k, s_{k+1}, P_1, \ldots, P_k, P_{k+1}, r_1, \ldots, r_k, r_{k+1}, 1).$$

      iii. $k \leftarrow k + 1$.

▶ **Theorem 17.** *Fix any adversary $\mathcal{A}$ for Π' that corrupts $\epsilon' R(\Pi'_{\mathcal{A}})$ of the messages of total volume at most $\epsilon \mathrm{CC}(\Pi'_{\mathcal{A}})$, for $\epsilon \leq \frac{\alpha}{bd \log d}$, where $\Pi'_{\mathcal{A}}$ denotes the protocol Π' executed with the adversary $\mathcal{A}$. Then there exists an adversary $\mathcal{D}$ for the protocol Π, that corrupts at most $\epsilon' R(\Pi_{\mathcal{D}}) + 2\epsilon' \log_b \mathrm{CC}(\Pi_{\mathcal{D}})$ messages of total volume at most $2\epsilon \mathrm{CC}(\Pi_{\mathcal{D}})$,[14] such that the following holds:*

1. *$\Pi'_{\mathcal{A}}$ always sends at least $t_{\min}$ bits.*
2. *$\mathrm{CC}(\Pi'_{\mathcal{A}}) \leq (1 + 2600C\alpha)\,\mathrm{CC}(\Pi_{\mathcal{D}})$.*
3. *$R(\Pi'_{\mathcal{A}}) \leq R(\Pi_{\mathcal{D}}) + 2\log_b \mathrm{CC}(\Pi_{\mathcal{D}})$.*
4. *When $\Pi'_{\mathcal{A}}$ ends, both Alice and Bob (separately) can efficiently compute their view of the transcript of $\Pi_{\mathcal{D}}$.*
5. *The adversary $\mathcal{D}$ chooses the hash collisions in a probabilistic manner such that for every t and every r, with probability $\geq 1 - 20 \cdot 2^{-\frac{\gamma}{3d}t}$, the volume of hash collisions in the first t bits of $\Pi_{\mathcal{D}}$ is at most $35\gamma t$, and with probability $\geq 1 - 80r \cdot 2^{-\frac{7\gamma^8}{d}r}$, the number of rounds with hash collisions in the first r rounds of $\Pi_{\mathcal{D}}$ is at most $100\gamma r$.*
6. *Π' is efficiently computable if Π is efficiently computable.*

The proof of Theorem 17 is deferred to full version.

7 Putting it all Together

In this section, we prove our main theorem (Theorem 4), using the theorems from previous sections. We restate our main theorem for the sake of convenience.

▶ **Theorem 18.** *There exists a universal constant $\alpha_0 \geq 0$ such that for any blowup parameters $\alpha \leq \alpha_0$ and $\alpha' \leq 1$, there exist parameters $\epsilon = \tilde{\Omega}\left(\alpha^{3+\frac{1}{\alpha'}}\right)$, $\epsilon' = \tilde{\Omega}(\alpha\alpha'^3)$, and $\delta = \alpha^{O(1/\alpha')}$, and there exists a probabilistic oracle machine S, such that for any protocol $\Pi = (A, B)$,*

[13] The binary representation of $|T|'$ has length $\leq b^k W$ since $b^k W \geq \frac{1}{\alpha}$ and so $\log |T|' \leq \log \frac{b^k W}{\alpha} \leq b^k W$.
[14] $\Pi_{\mathcal{D}}$ denotes the protocol Π executed with the adversary $\mathcal{D}$.

in which the parties always transmit at least $t_{\min}$ bits (even in the presence of error), and for any adversary $\mathcal{A}$ that corrupts at most ϵ-fraction of the bits of the simulated protocol $\Pi' = (S^A, S^B)$, the protocol $\Pi'_{\mathcal{A}}$ (which is the protocol Π' executed with the adversary $\mathcal{A}$), satisfies the following properties.

1. $\mathrm{CC}(\Pi'_{\mathcal{A}}) \geq t_{\min}$.
2. *There exists* $t_0 = (1 + \tilde{O}(\alpha))\mathrm{CC}(A, B)$ *such that for all* $t > t_0$

$$\Pr[\mathrm{CC}(\Pi'_{\mathcal{A}}) > t] \leq 2 \cdot 2^{-\delta t} ,$$

   where the probability over the private randomness of S.
3. *There exists* $r_0 = (1 + O(\alpha'))\, R(A, B) + O\left(\frac{1}{\log \frac{2}{\alpha'}} \log \mathrm{CC}(A, B) + 1\right)$ *such that for any $r \geq r_0$, if at most ϵ'-fraction of the messages are α^2-corrupted, then*

$$\Pr\left[R(\Pi'_{\mathcal{A}}) > r\right] \leq 2 \cdot 2^{-\delta r} ,$$

   where the probability over the private randomness of S.
4. *For any $t > 0$,*

$$\Pr\left[(\mathsf{Output}(\Pi'_{\mathcal{A}}) \neq \mathsf{Trans}(\Pi)) \ \wedge \ (\mathrm{CC}(\Pi'_{\mathcal{A}}) > t)\right] \leq 2 \cdot 2^{-\delta t} ,$$

   where the probability over the private randomness of S.
5. *S is a probabilistic polynomial time oracle machine, and hence the computational efficiency of S^A and S^B is comparable to that of A and B, respectively.*

In the proof of this theorem, we use an error correcting code from a recent work of Guruswami and Li [18].

▶ **Theorem 19** ([18]). *For every $\alpha > 0$ there is an explicit encoding scheme* $\mathrm{Enc}, \mathrm{Dec} : \{0,1\}^* \to \{0,1\}^*$ *with the following properties:*

1. *For any $m \in \{0,1\}^*$ we have* $|\mathrm{Enc}(m)| = (1 + \tilde{O}(\alpha))|m|$.
2. *For any $m \in \{0,1\}^*$ and any y that can be obtained from* $\mathrm{Enc}(m)$ *by* $\alpha^2 \cdot |\mathrm{Enc}(m)|$ *insertions and deletions,* $\mathrm{Dec}(y) = m$.
3. Enc *and* Dec *are computable by a polynomial time Turing machine.*

In the proof of Theorem 18 we use the following padding claim.

▷ Claim 20. Let $\alpha, \beta \leq 0.1$, and $L_0 \geq \alpha^{-1}$. Then any $(\alpha, 2\beta)$-smooth protocol Π can be efficiently converted into an (α, β)-smooth protocol Π' such that

- Π can be computed from the first $(1 - 2\alpha)$ fraction of bits of Π'.
- $\mathrm{CC}(\Pi) + L_0 \leq \mathrm{CC}(\Pi') \leq (1 + 13\alpha)\mathrm{CC}(\Pi) + 3L_0$.
- $R(\Pi') \leq R(\Pi) + \log_{\frac{1}{\beta}} \mathrm{CC}(\Pi) + \log_{\frac{1}{\beta}} L_0 + 1$.

The proof of this claim is deferred to full version.

Proof of Theorem 18. Fix any $\alpha \leq \alpha_0$ and $\alpha' \leq 1$. Let C be the constant from Lemma 13 (see Section 5). Let Enc, Dec be the encoding scheme from Theorem 19 with the parameter α. Recall that for all m, $|\mathrm{Enc}(m)| = (1 + \tilde{O}(\alpha))|m|$. Let α_1 be the maximal constant that satisfies,

$$\forall m : |\mathrm{Enc}(m)| \leq 2|m|. \tag{6}$$

We define $\alpha_0 = \min\{\alpha_1, \frac{1}{3200C}\}$. Given α, α' define,

$$\beta = \frac{\alpha^{\frac{1}{\alpha'}}}{320 \log^2 \frac{1}{\alpha}} , \ \gamma = \frac{\alpha^{\frac{4}{\alpha'}}}{2^{40}} , \ b = \frac{2}{\alpha'} , \ d = \frac{\log \frac{1}{\gamma}}{\alpha} , \ L_0 = 250C\alpha^{-2} \log d ,$$

$$\epsilon = \min\left\{\frac{\alpha^3}{2bd\log d}, \frac{\alpha^3\beta}{320}\right\} \ , \ \epsilon' = \frac{\alpha{\alpha'}^3}{\log^3\frac{1}{\alpha}}, \text{ and } \delta = \gamma^9 \ .$$

These parameters were chosen to satisfy the following claim.

▷ Claim 21. Our parameters satisfy the following:[15]

1. $\epsilon = \tilde{\Omega}(\alpha^{3+\frac{1}{\alpha'}})$, $\epsilon' = \tilde{\Omega}(\alpha{\alpha'}^3)$ and $\delta = \alpha^{O(1/\alpha')}$.
2. $\alpha < \frac{1}{4}$ and $\beta \leq \frac{\alpha}{16}$.
3. $\alpha, \beta < 0.1$ and $L_0 \geq \alpha^{-1}$.
4. $\alpha \leq 0.01$, $\alpha' \leq 1$, $d \geq \frac{1}{\alpha}$ and $\beta \leq \min\left\{\alpha^{\frac{1}{\alpha'}}, \frac{1}{5\alpha d^2}\right\}$.
5. $\gamma \leq \min\left\{\frac{1}{d}, 2^{-b}\right\}$, $d \geq \frac{\log\frac{1}{\gamma}}{\alpha}$, and $L_0 \geq \max\{\alpha^{-2}, 250C\alpha^{-1}\log d\}$.
6. $\frac{2\epsilon}{\alpha^2} \leq \frac{\alpha}{bd\log d}$.
7. $18\beta^{-1}(35\gamma + \frac{4\epsilon}{\alpha^2}) + 20d\beta^{-1}(35\gamma + 4\epsilon) \leq \alpha$.
8. $(100\gamma + \epsilon') \cdot 906d\log\frac{1}{\beta} \leq \alpha'$ and $1812d\log\frac{1}{\beta}\epsilon' \leq \frac{1}{\log\frac{2}{\alpha'}}$.
9. $\delta \leq \frac{1}{(10\alpha^{-1}+10)L_0}$, and for all $x > 0$ we have that

$$2\cdot 2^{-\delta x} \geq \min\left\{1, \frac{120d}{\gamma}\cdot 2^{-\frac{\gamma}{3d}x} + \gamma^{-17}\cdot 2^{-\frac{3}{2}\gamma^8 x}\right\} \ .$$

The protocol Π' is defined as follows:

1. Convert Π into a $(\alpha, 2\beta)$-smooth protocol Π_{smooth} by applying Lemma 6 (Section 3) to Π, with respect to parameters $(\alpha, 2\beta)$. These parameters satisfy the requirements in Lemma 6 by Item 2.
2. Convert Π_{smooth} into Π_{pad} using Claim 20 (above) with parameters α, β, L_0. These parameters satisfy the requirements in Claim 20 by Item 3.
3. Convert Π_{pad} to the error-resilient protocol Π_{ideal}, which is error-resilient in the ideal hash model, by applying the protocol from Section 4 to Π_{pad}, with parameters $\alpha, \alpha', \beta, d$. Jumping ahead, note that by Item 4, these parameters satisfy Equation (3) which is required in order to apply Theorem 9.
4. Convert Π_{ideal} to the protocol Π_{rand}, which is obtained by instantiating the ideal hash using private randomness, obtained by applying the protocol described in Section 6 to Π_{ideal}. Jumping ahead, note that by Items 5 and 6, imply that Equations (4) and (5) are satisfied and the requirements of Theorem 17 are satisfied with respect to any adversary $\mathcal{A}$ that corrupts messages of total volume $\leq \frac{2\epsilon}{\alpha^2}\mathrm{CC}((\Pi_{\mathsf{rand}})_{\mathcal{A}})$.
5. Convert Π_{rand} to $\Pi' = (S^A, S^B)$, where Π' is the same as Π_{rand}, except that each message is sent encoded with the error correcting code from Theorem 19 with parameter α.

▶ **Lemma 22.** *The protocol $\Pi' = (S^A, S^B)$ satisfies the conditions of Theorem 18.*

The proof of Lemma 22 is deferred to full version. ◀

[15] Each of the following items will later be used to apply a different theorem from previous sections.

References

1 Shweta Agrawal, Ran Gelles, and Amit Sahai. Adaptive protocols for interactive communication. In *IEEE International Symposium on Information Theory, ISIT 2016, Barcelona, Spain, July 10-15, 2016*, pages 595–599, 2016. doi:10.1109/ISIT.2016.7541368.

2 Zvika Brakerski and Yael Tauman 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*, pages 160–166, 2012. doi:10.1109/FOCS.2012.22.

3 Zvika Brakerski, Yael Tauman Kalai, and Moni Naor. Fast Interactive Coding against Adversarial Noise. *J. ACM*, 61(6):35:1–35:30, 2014. doi:10.1145/2661628.

4 Zvika Brakerski and Moni Naor. Fast Algorithms for Interactive Coding. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, pages 443–456, 2013.

5 Mark Braverman. Towards deterministic tree code constructions. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 161–167, 2012. doi:10.1145/2090236.2090250.

6 Mark Braverman and Klim Efremenko. List and Unique Coding for Interactive Communication in the Presence of Adversarial Noise. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, FOCS '14, pages 236–245, 2014.

7 Mark Braverman, Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Constant-rate coding for multiparty interactive communication is impossible. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 999–1010, 2016. doi:10.1145/2897518.2897563.

8 Mark Braverman, Ran Gelles, Jieming Mao, and Rafail Ostrovsky. Coding for Interactive Communication Correcting Insertions and Deletions. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 61:1–61:14, 2016. doi:10.4230/LIPIcs.ICALP.2016.61.

9 Mark Braverman and Anup Rao. Towards coding for maximum errors in interactive communication. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 159–166, New York, NY, USA, 2011. ACM. doi:10.1145/1993636.1993659.

10 Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Maximal Noise in Interactive Communication Over Erasure Channels and Channels With Feedback. *IEEE Trans. Information Theory*, 62(8):4575–4588, 2016. doi:10.1109/TIT.2016.2582176.

11 Ran Gelles. Coding for Interactive Communication: A Survey. *Foundations and Trends in Theoretical Computer Science*, 13(1-2):1–157, 2017. doi:10.1561/0400000079.

12 Ran Gelles and Bernhard. Capacity of Interactive Communication over Erasure Channels and Channels with Feedback. *SIAM J. Comput.*, 46(4):1449–1472, 2017. doi:10.1137/15M1052202.

13 Ran Gelles, Bernhard Haeupler, Gillat Kol, Noga Ron-Zewi, and Avi Wigderson. Towards Optimal Deterministic Coding for Interactive Communication. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1922–1936, 2016. doi:10.1137/1.9781611974331.ch135.

14 Ran Gelles and Yael Tauman Kalai. Constant-Rate Interactive Coding Is Impossible, Even In Constant-Degree Networks. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:95, 2017. URL: https://eccc.weizmann.ac.il/report/2017/095.

15 Ran Gelles, Ankur Moitra, and Amit Sahai. Efficient and Explicit Coding for Interactive Communication. In *Proceeding of the IEEE Symposium on Foundations of Computer Science*, FOCS '11, pages 768–777, 2011.

16 Mohsen Ghaffari and Bernhard Haeupler. Optimal Error Rates for Interactive Coding II: Efficiency and List Decoding. *CoRR*, abs/1312.1763, 2013. arXiv:1312.1763.

17 Mohsen Ghaffari, Bernhard Haeupler, and Madhu 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*, pages 794–803, 2014. doi:10.1145/2591796.2591872.

18 Venkatesan Guruswami and Ray Li. Efficiently decodable insertion/deletion codes for high-noise and high-rate regimes. In *IEEE International Symposium on Information Theory, ISIT 2016, Barcelona, Spain, July 10-15, 2016*, pages 620–624, 2016. doi:10.1109/ISIT.2016.7541373.

19 Bernhard Haeupler. Interactive Channel Capacity Revisited. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 226–235, 2014. doi:10.1109/FOCS.2014.32.

20 Bernhard Haeupler and Amirbehshad Shahrasbi. Synchronization strings: codes for insertions and deletions approaching the Singleton bound. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 33–46, 2017. doi:10.1145/3055399.3055498.

21 Bernhard Haeupler, Amirbehshad Shahrasbi, and Ellen Vitercik. Synchronization Strings: Channel Simulations and Interactive Coding for Insertions and Deletions. *CoRR*, abs/1707.04233, 2017. arXiv:1707.04233.

22 Bernhard Haeupler and Ameya Velingker. Bridging the Capacity Gap Between Interactive and One-Way Communication. *CoRR*, abs/1605.08792, 2016. arXiv:1605.08792.

23 Abhishek Jain, Yael Tauman Kalai, and Allison Lewko. Interactive Coding for Multiparty Protocols. In *Proceedings of the 6th Conference on Innovations in Theoretical Computer Science, ITCS '15*, pages 1–10, 2015.

24 Gillat Kol and Ran Raz. Interactive channel capacity. In *STOC '13: Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, pages 715–724, New York, NY, USA, 2013. ACM. doi:10.1145/2488608.2488699.

25 Joseph Naor and Moni Naor. Small-Bias Probability Spaces: Efficient Constructions and Applications. *SIAM J. Comput.*, 22(4):838–856, 1993. doi:10.1137/0222053.

26 Sridhar Rajagopalan and Leonard Schulman. A coding theorem for distributed computation. In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 790–799, New York, NY, USA, 1994. ACM. doi:10.1145/195058.195462.

27 Leonard J. Schulman. Communication on noisy channels: a coding theorem for computation. *Foundations of Computer Science, Annual IEEE Symposium on*, pages 724–733, 1992. doi:10.1109/SFCS.1992.267778.

28 Leonard J. Schulman. Deterministic coding for interactive communication. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 747–756, New York, NY, USA, 1993. ACM. doi:10.1145/167088.167279.

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

30 Claude E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001. Originally appeared in *Bell System Tech. J.* 27:379–423, 623–656, 1948.

31 Alexander A. Sherstov and Pei Wu. Optimal Interactive Coding for Insertions, Deletions, and Substitutions. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:79, 2017. URL: https://eccc.weizmann.ac.il/report/2017/079.