2014 IEEE Annual Symposium on Foundations of Computer Science

Interactive Channel Capacity Revisited

Bernhard Haeupler
Microsoft Research
haeupler@cs.cmu.edu

***Abstract*—We provide the first capacity approaching coding schemes that robustly simulate any interactive protocol over an *adversarial channel* that corrupts any ϵ fraction of the transmitted symbols. Our coding schemes achieve a communication rate of $1 - O(\sqrt{\epsilon \log\log 1/\epsilon})$ over any adversarial channel. This can be improved to $1 - O(\sqrt{\epsilon})$ for random, oblivious, and computationally bounded channels, or if parties have shared randomness unknown to the channel.**

Surprisingly, these rates exceed the $1 - \Omega(\sqrt{H(\epsilon)}) = 1 - \Omega(\sqrt{\epsilon \log 1/\epsilon})$ interactive channel capacity bound which [Kol and Raz; STOC'13] recently proved for *random errors*. We conjecture $1 - \Theta(\sqrt{\epsilon \log\log 1/\epsilon})$ and $1 - \Theta(\sqrt{\epsilon})$ to be the *optimal* rates for their respective settings and therefore to capture the *interactive channel capacity* for *random and adversarial errors*.

In addition to being very communication efficient, our randomized coding schemes have multiple other advantages. They are computationally efficient, extremely natural, and significantly simpler than prior (non-capacity approaching) schemes. In particular, our protocols do not employ any coding but allow the original protocol to be performed *as-is*, interspersed only by short exchanges of hash values. When hash values do not match, the parties backtrack. Our approach is, as we feel, by far the simplest and most natural explanation for why and how robust interactive communication in a noisy environment is possible.

I. Introduction

We study the *interactive channel capacity* of random and adversarial error channels, that is, the fundamental limit on the communication rate up to which any interactive communication can be performed in the presence of noise. We give novel coding schemes which, for a wide variety of channels, achieve and resolve the corresponding interactive channel capacity up to a constant in the second order term for any small error rate ϵ. Our coding schemes are extremely simple, computationally efficient, and give the most natural and intuitive explanation for why and how error correction can be performed in interactive communications.

A. Prior Work

From Error Correcting Codes . . .: The concept of (forward) error correcting codes has fundamentally transformed the way information is communicated and stored and has had profound and deep connections in many sub-fields of mathematics, engineering, and beyond. Error correcting codes allow to add redundancy to any message consisting of n symbols, e.g., bits, and transform it into a coded message with $\alpha n + o(n)$ symbols from a finite alphabet Σ from which one can recover the original message even if any ϵ-fraction of the symbols are corrupted in an arbitrary way. This can be used to store and recover information in a fault tolerant way (e.g., in CDs, RAM, . . .) and also leads to robust transmissions over any noisy channel. In the later case, one denotes with $R = \frac{1}{\alpha \log |\Sigma|}$ the *communication rate* at which such a communication can be performed with negligible probability of failure and for a given channel one denotes with the *channel capacity* C the supremum of the achievable rates for large n.

The groundbreaking works of Shannon and Hamming showed that the capacity C of any binary channel with an ϵ fraction of noise satisfies $C = 1 - \Theta(H(\epsilon))$, where $H(\epsilon) = \epsilon \log \frac{1}{\epsilon} + (1-\epsilon) \log \frac{1}{1-\epsilon}$ denotes the binary entropy function which behaves like $H(\epsilon) = \epsilon \log \frac{1}{\epsilon} + O(\epsilon)$ for $\epsilon \to 0$. More precisely, for random errors as modeled by the *binary symmetric channel* (BSC), which flips each transmitted bit with probability ϵ, Shannon's celebrated theorem states that the rate $R = 1 - H(\epsilon)$ is the exact asymptotic upper and lower bound for achieving reliable communication. Furthermore, for arbitrarily, i.e., adversarially, distributed errors Hamming's work shows that the rate $R = 1 - \Theta(H(\epsilon))$ remains achievable. Determining the optimal rate of a binary code or even just the constant hidden in the second order term is a fundamental and widely open question in coding theory. The popular guess or conjecture is that the Gilbert-Varshamov bound of $R \leq 1 - H(2\epsilon)$ is essentially tight.

. . . to Coding Schemes for Interactive Communications: These results apply to *one-way* communications in which one party, say Alice, wants to communicate information to another party, say Bob, in the presence of noise. In this paper we are interested in the same concept but for settings in which Alice and Bob have a *two-way* or *interactive* communication which they want to make robust to noise by adding redundancy. More precisely, Alice and Bob have some conversation in mind which in a noise-free environment can be performed by exchanging n symbols in total. They want a *coding scheme* which adds redundancy and transforms any such conversation into a slightly longer αn-symbol conversation from which both parties can recover the original conversation outcome even when any ϵ fraction of the coded conversation is corrupted.

The reason why one cannot simply use error correcting codes for each message is that misunderstanding just one message, which corresponds only to a $1/n$ fraction of corruptions for an n message interaction, leads to the remainder of the conversation becoming irrelevant and useless. It is therefore a priori not clear that tolerating some (even tiny) constant fraction of errors ϵ is even possible.

In 1993 Schulman [15] was the first to address this question. In his beautiful and at the time surprising work he showed that tolerating an $\epsilon = 1/240$ fraction of the adversarial errors

0272-5428/14 $31.00 © 2014 IEEE
DOI 10.1109/FOCS.2014.32

is possible for some constant overhead $\alpha = \Theta(1)$. This also directly implies that any error rate bounded away by a constant from $1/2$ can be tolerated for the easier random errors setting, since one can easily reduce the error rate by repeating symbols multiple times. Later, Braverman and Rao [5] showed that an adversarial error rate of up to $\epsilon < 1/4$ could be tolerated with a constant overhead $\alpha = \Theta(1)$. Lastly, [4], [6], [7], [10], [11] determined the full error rate region in which a non-zero communication rate is possible in a variety of settings. While the initial coding schemes were not computationally efficient, because they relied on powerful but hard to construct tree codes, later results [1], [2], [9], [10] provided polynomial time schemes using randomization.

Capacity Approaching Coding Schemes: None of these works considers the communication rate that can be maintained for a given noise level ϵ. In fact, each of these schemes has a relatively large unspecified constant factor overhead α even for negligible amounts of noise $\epsilon \to 0$. This is unsatisfactory as one would expect the necessary amount of redundancy to vanish in this case. However, capacity approaching schemes were considered out of scope of techniques up to this point [3, Problem 8]. The work by Kol and Raz [13] was the first to consider the communication rate up to which interactive communication could be maintained over a noisy channel. In particular, they studied random errors, as modeled by the BSC, with an error probability $\epsilon \to 0$. Under seemingly reasonable assumptions they proved an upper bound of $1 - \Omega(\sqrt{H(\epsilon)})$, that is, that some protocols cannot be robustly simulated with a rate exceeding $1 - \Omega(\sqrt{H(\epsilon)})$. This is significantly lower than the $1 - H(\epsilon)$ bound for the standard one-way setting. They also gave a coding scheme that achieves a matching rate of $1 - O(\sqrt{H(\epsilon)})$ for the same setting. This was seen as a characterizing the BSC *interactive channel capacity* up to constants in the second order term, a notable breakthrough.

B. Our Results

Coding Schemes for Adversarial Errors: This work started out as an attempt to address the question of communication rate and interactive channel capacity in the much harsher adversarial noise setting. In particular, the hope was to design a *capacity approaching* coding scheme for adversarial errors that could achieve a communication rate approaching one as the noise level ϵ goes to zero.

To our shock the communication rates of our final coding schemes exceed the $1-\Omega(\sqrt{H(\epsilon)}) = 1-\Omega(\sqrt{\epsilon \log \frac{1}{\epsilon}})$ bound of [13] even though they work in the much harder adversarial noise setting. In particular, the new communication schemes operate at a communication rate of $1 - O(\sqrt{\epsilon})$ against any worst-case oblivious channel and even channels controlled by a fully adaptive adversary as long as this adversary is computationally efficient or as long as the parties share some randomness unknown to the adversary. These trivially include the case of i.i.d. random errors. For the unrestricted fully adaptive adversarial channel we achieve a rate of $1 - O(\sqrt{\epsilon \log \log \frac{1}{\epsilon}})$ which is still lower than the bound of the impossibility result of [13].

Interactive Channel Capacity Revisited: We uncover that these differences stem from important but subtle variations in the assumptions on the *communication order*, that is, the order in which which Alice and Bob speak, both for the original (noiseless) input protocol Π and for its simulation Π':

The standard setting, used by all works prior to [13], assumes that Π is alternating, that is, that the parties take turns sending one symbol each. Both our $1 - O(\sqrt{\epsilon})$-rate and our $1 - O(\sqrt{\epsilon \log \log \frac{1}{\epsilon}})$-rate coding scheme as well as the $1-O(\sqrt{H(\epsilon)})$-rate coding scheme of [13] work in this setting. For all three protocols the simulation Π' can furthermore be chosen to have the same fixed alternating communication order. We remark that if one does not care about constant factors in the communication rate assuming an alternating input protocol is essentially without loss of generality because any protocol can be transformed to be alternating while only increasing its length by at most a factor of two. However, this transformation and wlog-assumption cannot be applied to design capacity approaching protocols. Regardless, the setting of alternating protocols and simulations is a very simple and clean setting which still encapsulates the characteristics and challenges of the problem.

The impossibility result of [13] however does not apply to alternating protocols. Instead, an input protocol with a more complex communication order is assumed. More importantly, the simulations Π' are restricted to be non-adaptive, that is, have an a priori fixed communication order which defines for each time step which party sends and which listens. The $1-\Omega(\sqrt{\epsilon \log 1/\epsilon})$ lower bound of [13] subtly but nonetheless crucially builds on this non-adaptivity assumption.

After understanding these issues better, we point out that insisting on non-adaptive simulations Π' is too restrictive for general input protocols Π in a very strong sense: Many non-alternating input protocols Π simply cannot be simulated robustly without losing at least a constant factor in the communication rate. This impossibility is furthermore essentially unrelated to the type of noise in the channel and therefore unrelated to the question of interactive channel capacity. More precisely, we conjecture the following $1 - \Omega(1)$ impossibility result to hold:

Conjecture I.1. *Any protocol Π with a sufficiently non-regular, e.g., pseudo-random, communication order cannot be robustly simulated by any non-adaptive protocol Π' with a rate of $R = 1 - o(1)$. This is true for essentially any channel introducing some error, in particular, even for channels introducing merely a single random error or erasure.*

In this work we show that there is a natural way to circumvent this impossibility barrier without having to restrict the input protocols Π that can be simulated. In particular, our protocols very naturally simulate *any* general input protocol Π if the simulation Π' is allowed to be have an adaptive communication order, as introduced in [11].

We furthermore conjecture that the $1 - O(\sqrt{\epsilon})$ bound we present in this work is the natural and tight bound on the maximum rate that can be achieved in a wide variety of interactive communication settings:

Conjecture I.2. *The maximal rate achievable by an interactive coding scheme for any binary error channel with random or oblivious errors is $1 - \Theta(\sqrt{\epsilon})$ for a noise rate $\epsilon \to 0$. This also holds for for fully adversarial binary error channels if the adversary is computationally bounded or if parties have access to shared randomness that is unknown to the channel. It also remains true for all these settings regardless whether one restricts the input protocols to be alternating or not.*

We suspect this claim also extends to larger alphabets. We feel that the broadness and robustness of this bound might justify regarding $1-\Theta(\sqrt{\epsilon})$ as *the* interactive channel capacity of a random error channel, even though this paper clearly demonstrates how careful and precise one needs to specify how protocols can use a channel before being able to talk about rates and capacities.

For the fully adversarial with no shared randomness and a binary channel alphabet we conjecture our rate of $1 - O(\sqrt{\epsilon \log\log \frac{1}{\epsilon}})$ to be tight as well. This bound does not hold for larger alphabets (see [12]):

Conjecture I.3. *The interactive channel capacity for the fully adversarial binary error channels in the absence of shared randomness is $1 - \Theta\left(\sqrt{\epsilon \log\log \frac{1}{\epsilon}}\right)$ for a noise rate $\epsilon \to 0$.*

Lastly, we remark that subsequent to this work it was shown in [8] that a higher rate of $1 - \Theta(H(\epsilon))$ is possible for coding schemes that robustly simulate any (alternating) protocol over random or adversarial channels with feedback or over random or adversarial erasure channels.

Simple, Natural, Communication and Computationally Efficient Coding Schemes: In addition to being capacity approaching for worst case errors with an optimal asymptotic dependence on the noise level ϵ the new coding schemes also have the advantage of being much simpler and more natural than prior coding schemes. They are essentially the first schemes for adversarial errors that do not rely on tree-codes. In fact, they do not perform any coding at all. Instead, they operate along the following extremely natural template:

Template I.1 (Making a Conversation robust to Noise)**.** *Both parties have their original conversation as if there were no noise except that*

1) *sporadically a concise summary (an $\Theta(1)$ or $\Theta(\log\log n)$ bit random hash value) of the conversation up to this point is exchanged.*
2) *If the summaries match the conversation continues.*
3) *If the summaries do not match, because the noise caused a misunderstanding, then the parties backtrack.*

Some details go into how to compute the summaries and how to coordinate the backtracking steps. Still, the protocol stays extremely simple and this outline is so intuitive that it can be easily explained to non-experts. In that respect it can be seen as demystifying Schulman's result that interactive communication can be performed at a constant rate in the presence of a constant fraction of adversarial noise. Our proofs for why these constant or $\Theta(\log\log n)$ size hash values are sufficient (and necessary) are simple and completely elementary with the standard hash functions from [14] being the only non-trivial black-box used. Furthermore, since our alternative proof is based solely on hashing it directly leads to a computationally efficient "coding" scheme. This scheme works without any assumptions on the structure of the original protocol and is so simple that real-world use-cases and implementations become a possibility. In fact, Microsoft has a utility patent pending. All in all, we feel that this paper gives the simplest, most natural, and most intuitive explanation for why robust interactive communication in a noisy environment is possible and how it can be achieved.

C. Organization

The remainder of this paper is organized as follows. In Section II we provide preliminaries and the channel and interactive communication models. In Section III we explain the fundamental difference between error correction for interactive communications and the classical one-way setting. In particular, we explain why a communication rate of $1-\Omega(\sqrt{\epsilon})$ is the best one can hope for in an interactive setting. In Section IV we provide a simple coding scheme achieving this rate against adversarial errors if the channel operates on a logarithmic bit-size alphabet. In Section V we explain the barriers in extending this algorithm to a constant size alphabet and give an overview of the techniques we use to overcome them. In Section VI we then provide our new coding schemes and we use Section VII to prove their correctness.

II. Definitions and Preliminaries

A. Interactive Protocols and Communication Order

An *interactive protocol* Π defines some communication performed by two parties, Alice and Bob, over a channel with alphabet Σ. After both parties are given an input the protocol operates in n *rounds*. For each round of communication each party decides independently whether to listen or transmit a symbol from Σ based on its state, its input, its randomness, and the history of communication, i.e., the symbols received by it so far. All our protocols will utilize *private randomness* which is given to each party in form of its own infinite string of independent uniformly random bits. It is also interesting to consider settings with *shared randomness* in which both parties in every round i have access to the same infinite random bit-string R_i.

We call the order in which Alice and Bob speak or listen the *communication order* of a protocol. Prior works have often studied *non-adaptive* protocols for which this communication order is predetermined. In this case, which player transmits or listens depends only on the round number and it is deterministically ensured that exactly one party transmits in each round. If the such a non-adaptive communication order repeats itself

in regular intervals we call the protocol *periodic* and call the smallest length of such an interval the *period*. The simplest communication order has Alice and Bob take taking turns. We call such a protocol, with period two, *alternating*.

B. Adversarial and Random Communication Channels

The communication between the two parties goes over a *channel* which delivers a possibly corrupted version of the chosen symbol of a transmitting party to a listening party. In particular, if exactly one party listens and one transmits then the listening party receives the symbol chosen by the transmitting party unless the channel interferes and corrupts the symbol.

In the *fully adversarial channel* model with *error rate* ϵ the number of such interferences is at most ϵN for an N round protocol and the adversary chooses the received symbol arbitrarily. In particular, the adversary gets to know the length N of the protocol and therefore also how many corruptions it is allowed to introduce. In each round it can then decide whether to interfere or not and what to corrupt a transmission to based on its state, its own randomness, and the communication history observed by it, that is, all symbols sent or received by the parties so far. The adversary does not get to know or base its decision on the private randomness of Alice or Bob (except for what it can learn about it through the communicated symbols). In the shared randomness setting we differentiate between the default setting, in which the adversary gets to know the shared randomness as well, that is, base its decisions in round i also on any R_j with $j \leq i$, and the *hidden shared randomness* setting in which the the shared randomness is a secret between Alice and Bob which the adversary does not know.

We also consider various relaxations of this all powerful fully adversarial channel model: We call an adversary *computationally bounded* if the decisions of the adversary in each round are required to be computable in time polynomial in N. We call an adversary *oblivious* if it makes all its decisions in advance, in particular, independently of the communication history. A particular simple such oblivious adversary is the *random error channel* which introduces a corruption in each round independently with probability ϵ. We will consider random channels mostly for binary channels for which a corruption is simply a bit-flip.

C. Adaptive Interactive Protocols

It is natural and in many cases important to also allow for *adaptive* protocols which base their communication order decisions on the communication history. We follow [11] in formalizing the working of the channel in these situations. In particular, in the case of both parties transmitting no symbol is delivered to either party because neither party listens anyway. In the case of both parties listening the symbols received are undetermined with the requirement that the protocol works for any received symbols. In many cases it is easiest to think of the adversary being allowed to choose the received symbols, without it being counted as a corruption.

D. Robust Simulations

A protocol Π' is said to *robustly simulate* a deterministic protocol Π over a channel C if the following holds: Given any inputs to Π, both parties can (uniquely) decode the transcript of the execution of Π over the noise free channel on these inputs from the transcript of an execution of Π' over the channel C. For *randomized protocols* we say protocol Π' *robustly simulates* a protocol Π with *failure probability* p over a channel C if, for any input and any adversary behind C, the probability that both parties correctly decode is at least $1-p$. We note that the simulation Π' typically uses a larger number of rounds, e.g., αn rounds for some $\alpha > 1$.

III. Channel Capacities for Interactive vs. One-way Communication

In this section we explain the important difference in correcting errors for interactive communications in contrast to the standard one-way setting of error correcting codes. We then quantify this difference and give the high-level argument for why $1-\Omega(\sqrt{\epsilon})$ is the best possible communication rate one can expect for coding schemes that make interactive communications robust to even just random noise. The impossibility result of [13] can be seen as formalizing a very similar argument.

A. The Difficulties of Coding for Interactive Communications

We first explain, on a very intuitive level, why making interactive communications resilient to noise is much harder than doing the same for one-way communications. In particular, we want to contrast the task of Alice transmitting some information to Bob with the task of Alice and Bob having a conversation, e.g., an interview of Bob by Alice, both in a noisy environment. The main difference between the two tasks is that in the one-way communication Alice knows everything she wants to transmit a priori. This allows her to mix this information together in an arbitrary way by using redundant transmissions that protects everything equally well. This contrasts with an interactive communication where Alice's transmissions depend highly on the answers given by Bob. In our interview example, Alice cannot really ask the second question before knowing the answer to her first question as what she wants to know from Bob might highly depend on Bob's first answer. Even worse, Alice misunderstanding the first answer might completely derail the interview into a direction not related to the original (noise free) conversation. In this case, everything talked about in the continuing conversation will be useless, even without any further noise or misunderstandings, until this first misunderstanding is detected. Lastly, in order to resolve a detected misunderstanding the conversation has to backtrack all the way to where the misunderstanding happened and continue from there.

B. The $1-\Omega(\sqrt{\epsilon})$ Fundamental Rate Limit

Next, we aim to quantify this difference and argue for $1-\Theta(\sqrt{\epsilon})$ being a fundamental rate limit of interactive communication, even for random errors. We pick the simplest noise model and assume, for now, that Alice and Bob try

to communicate over a binary symmetric channel, that is, a binary random error channel which flips each bit transmitted independently with the small error probability ϵ. For the one-way communication task Alice can simply transmit everything she wants to send followed by a few check-sums over the complete message. It is a classical result that for a full error recovery it suffices to add to the transmission approximately $H(\epsilon)$ as many randomly picked linear check-sums as transmitted symbols, which in the binary case are simply parities over a randomly chosen subset. This leads to a rate of $1 - H(\epsilon)$ which is also optimal.

Now we consider Alice and Bob having an interactive conversation over the same channel. Because of the noise Alice and Bob will need to add some redundancy to their conversation at some point in order to at least detect whether a misunderstanding has happened. For this one (check-)bit is necessary. Say they do this after r steps. The likelihood for an error to have happened is $r\epsilon$ at this point and the length of the conversation that needs to be redone because of such a preceding misunderstanding is of expected length $\frac{r}{2}$. This leads to an expected rate loss of $\Theta(r\epsilon)$ because every r steps an expected $\frac{r^2\epsilon}{2}$ steps are wasted. With this reasoning one would like to make r as small as possible. However, adding one unit of redundancy every r steps leads to a rate loss of $\frac{1}{r}$ itself regardless of whether errors have occurred. Balancing r to minimize these two different causes of rate loss leads to an optimal rate loss of $\Theta(\min_r\{r\epsilon + 1/r\}) = \Theta(\sqrt{\epsilon})$ assuming r is set optimally to $r = \frac{1}{\sqrt{\epsilon}}$. This argument applies in essentially any interactive coding setting and explains why the fundamental channel capacity drops from $1 - H(\epsilon)$ to $1 - \Theta(\sqrt{\epsilon})$ for interactive communications.

IV. A Simple Coding Scheme for Large Alphabets

In this section, as a warmup, we give a simple coding scheme that achieves a rate of $1 - \Theta(\sqrt{\epsilon})$ against any fully adversarial error channel, albeit while assuming that the original protocol and the channel operate on words, that is, on the same $\Theta(\log n)$ bit-sized alphabet.

A. Overview

To design our coding scheme we follow the idea is outlined in Section I-B and turn the $1-\Omega(\sqrt{\epsilon})$ bound from Section III it into a converse, that is, prove it to be tight by designing a protocol achieving this rate. Our protocol can also be seen as a drastically simplified version of [1].

In particular, the parties start with performing the original interactive protocol without any coding for $r = \Theta(\frac{1}{\sqrt{\epsilon}})$ steps as if no noise is present at all. Both parties then try to verify whether an error has occurred. They do this by randomly sampling a hash function and sending both the description of the hash function as well as the hash value of their complete communication transcript up to this point. They use $r_c = O(1)$ symbols for this check and initiate a backtracking step if a mismatch is detected.

The (standard) hash functions used for this and throughout this paper are derived from the ϵ-biased probability spaces constructed in [14]. These hash functions give the following guarantees:

Lemma IV.1 ([14]). *For any n, any alphabet Σ, and any probability $0 < p < 1$, there exist $s = \Theta(\log(n \log |\Sigma|) + \log \frac{1}{p})$, $o = \Theta(\log \frac{1}{p})$, and a simple function h, which given an s-bit uniformly random seed S maps any string over Σ of length at most n into an o-bit output, such that the collision probability of any two n-symbol strings over Σ is at most p. In short:* $\forall n, \Sigma, 0 < p < 1 : \exists s = \Theta(\log(n \log |\Sigma|)+\log \frac{1}{p}), o = \Theta(\log \frac{1}{p}), h : \{0,1\}^s \times \Sigma^n \mapsto \{0,1\}^o \; s.t. \; \forall X, Y \in \Sigma^{\leq n}, X \neq Y, S \in \{0,1\}^s \; iid \; Bernoulli(1/2) : \quad P[h_S(X) = h_S(Y)] \leq p$

The setting of interest to our first algorithm is that any two $\Theta(n)$ bit strings can, with high probability, be distinguished by a random hash function which creates $o = \Theta(\log n)$ size fingerprints and requires a seed of only $\Theta(\log n)$ bits. This allows us to communicate both the selected hash function (seed) and the corresponding hash value of the $O(n)$ long transcript using only $r_c = \Theta(1)$ symbols of $\Theta(\log n)$ bits each.

A last minor technical detail is a simple preprocessing step in which the original protocol Π is modified by adding $\Theta(\sqrt{\epsilon}n)$ steps at the end, in which both parties send each other a fixed symbol. These steps should be interpreted as confirmation steps which reaffirm the correctness of the previous conversation. This ensures that the simulation Π' never runs out of steps of Π to simulate.

B. Algorithm

Putting these ideas together leads to our first, simple coding scheme which achieves the optimal $1 - \Theta(\sqrt{\epsilon})$ error rate for any logarithmic bit-sized alphabet:

Algorithm 1 Simple Coding Scheme for $\Theta(\log n)$-bit alphabet Σ

$\Pi \leftarrow n$-round protocol to be simulated + final confirmation steps
$hash \leftarrow$ hash family with $p = 1/n^5$ and $o = s = \Theta(\log n)$
$r_c \leftarrow \Theta(1)$; $r \leftarrow \left\lceil \sqrt{\frac{r_c}{\epsilon}} \right\rceil$; $R_{total} \leftarrow \lceil n/r + 32n\epsilon \rceil$; $\mathtt{T} \leftarrow \emptyset$
for $R = 1$ to R_{total} **do**
  $\mathtt{S} \leftarrow s$ uniformly random bits ▷ **Verification Phase**
  Send $(\mathtt{S}, hash_{\mathtt{S}}(\mathtt{T}), |\mathtt{T}|)$; Receive $(\mathtt{S}', H'_{\mathtt{T}}, l')$
  $H_{\mathtt{T}} \leftarrow hash_{\mathtt{S}'}(\mathtt{T})$; $l \leftarrow |\mathtt{T}|$
  if $H_{\mathtt{T}} = H'_{\mathtt{T}}$ **then** ▷ **Computation Phase**
    continue computation of Π for r communications and record those in $\mathtt{T}$
  else
    do r dummy communications keeping $\mathtt{T}$ unchanged
  if $H_{\mathtt{T}} \neq H'_{\mathtt{T}}$ and $l \geq l'$ **then** ▷ **Transition Phase**
    rollback computation of Π and transcript $\mathtt{T}$ by r steps
Output the outcome of Π corresponding to transcript $\mathtt{T}$

C. Proof of Correctness

Theorem IV.2. *Suppose any n-round protocol Π using an alphabet Σ of bit-size $\Theta(\log n)$. Algorithm 1 is a computationally efficient randomized coding scheme which given Π, with probability $1 - 2^{-\Theta(n\epsilon)}$, robustly simulates it over any fully adversarial error channel with alphabet Σ and error rate*

ϵ. *The simulation uses* $n(1+\Theta(\sqrt{\epsilon}))$ *rounds and therefore achieves a communication rate of* $1-\Theta(\sqrt{\epsilon})$.

Proof Outline: We show that the algorithm terminates correctly by defining an appropriate potential Φ. We prove that any iteration without an error or hash collision increases the potential by at least one while any error or hash collision reduces the potential by at most some fixed constant. Lastly, we show that with very high probability the number of hash collisions is at most $O(\epsilon n)$ and therefore negligible. This guarantees an overall potential increase that suffices to show that the algorithm terminates correctly after the fixed R_{total} number of iterations.

Potential: The potential Φ is based on the transcript $\mathtt{T}$ of both parties. We use $\mathtt{T}_A$ and $\mathtt{T}_B$ to denote the transcript of Alice and Bob respectively. We first define the following intermediate quantities: The agreement l^+ between the two transcripts at Alice and Bob is the number of blocks of length r in which they agree, that is, $l^+ = \left\lfloor \frac{1}{r} \max\{l' \text{ s.th. } \mathtt{T}_A[1,l'] = \mathtt{T}_B[1,l']\} \right\rfloor$. Similarly, we define the amount of disagreement l^- as the number of blocks they do not agree on: $l^- = \frac{|\mathtt{T}_A|+|\mathtt{T}_B|}{r} - 2l^+$. The potential Φ is now simply defined as $\Phi = l^+ - l^-$.

Corollary IV.3. *Each iteration of Algorithm 1 without a hash collision or error increases the potential* Φ *by at least one.*

Proof. If $\mathtt{T}_A = \mathtt{T}_B$ then both parties continue computing Π from the same place and, since no error happens, both parties correctly add the next r communications of Π to their transcripts. This increases l^+ and therefore also the overall potential Φ by one.

If $\mathtt{T}_A \neq \mathtt{T}_B$ and no hash collision happens then both parties realize this discrepancy and also learn the correct length of the other party's transcript. If $|\mathtt{T}_A| = |\mathtt{T}_B|$ then both parties backtrack one block which reduces l^- by two and thus increases the potential by two. Otherwise, the party with the longer transcript backtracks one block while the other party does not change its transcript. This reduces l^- by one and increases the overall potential Φ by one. □

Corollary IV.4. *Each iterations of Algorithm 1, regardless of the number of hash collisions and errors, decreases the potential* Φ *by at most three.*

Proof. No matter what is received during an iteration a party never removes more than one block from its transcript. Similarly, at most one block is added to $\mathtt{T}_A$ and $\mathtt{T}_B$. Overall in one iteration this changes l^+ by at most by one and l^- at most by two. The overall potential Φ changes therefore at most by three in any iteration. □

Next, we argue that the number of iterations of Algorithm 1 with a hash collision is negligible. To be precise, we say an iteration *suffers a hash collision* if $\mathtt{T}_A \neq \mathtt{T}_B$ but either $hash_{\mathtt{S}_B}(\mathtt{T}_A) = hash_{\mathtt{S}_B}(\mathtt{T}_B)$ or $hash_{\mathtt{S}_A}(\mathtt{T}_A) = hash_{\mathtt{S}_A}(\mathtt{T}_B)$. In particular, we do not count iterations as suffering a hash collision if the random hash functions sampled would reveal a discrepancy but this detection, e.g., in Line 12, is prevented by corruptions in the transmission of a hash value or seed. To prove the number of hash collisions to be small we crucially exploit the fact that the randomness used for hashing is sampled afresh in every iteration. In particular, it is sampled after everything that is hashed in this iteration is already irrevocably fixed. This independence allows to use the collision resistance property of Lemma IV.1 which shows that any iteration suffers from a hash collision with probability at most $p = 1/n^5$. A union bound over all iterations then shows that with high probability no hash collision happens at all. Furthermore, using the independence between iterations in combination with a standard tail bound proves that the probability of having a number of hash collisions of the same order of magnitude as the number of errors is at least $1-2^{-\Theta(\epsilon n)}$:

Corollary IV.5. *The number of iterations of Algorithm 1 suffering from a hash collision is at most* $6n\epsilon$ *with probability at least* $1-2^{-\Theta(\epsilon n)}$.

We are now ready to prove Theorem IV.2:

Proof of Theorem IV.2. There are at most $2n\epsilon$ errors and according to Corollary IV.5 at most $6n\epsilon$ iterations with a hash collision. This results in at most $8n\epsilon$ iterations in which, according to Corollary IV.4 the potential Φ decreases (by at most three). For the remaining $R_{total} - 8n\epsilon = \lceil n/r + 24n\epsilon \rceil$ iterations Corollary IV.3 shows that the potential Φ increases by one. This leads to a total potential of at least $\lceil n/r \rceil$ which implies that after the last iteration both parties agree upon the first n symbols of the execution of Π. This leads to both parties outputing the correct outcome and therefore to Corollary IV.5 being a correct robust simulation of Π.

To analyze the round complexity and communication rate we note that each of the R_{total} iteration consists of r computation steps and $r_c = \Theta(1)$ symbols exchanged during any verification phase. The total round complexity of Algorithm 1 is therefore $R_{total} \cdot (r + r_c) = \lceil n/r + 6n\epsilon \rceil \cdot (r + \Theta(1)) = n + \Theta(n\epsilon r + n/r + n\epsilon) = n(1 + \Theta(\epsilon r + 1/r))$. Choosing the optimal value of $r = \Theta(\frac{1}{\sqrt{\epsilon}})$, as done in Algorithm 1, leads to $n(1+\Theta(\sqrt{\epsilon})$ rounds in total, as claimed. □

V. Problems and Solutions for Small Alphabets

In this section we explain the barriers preventing Algorithm 1 to be applied to channels with small alphabets and then explain the solutions and ideas put forward in this work to circumvent them.

A. Problems with Small Alphabets

It is easy to see that the only thing that prevents Algorithm 1 from working over a smaller alphabet is that the verification phase uses $\Theta(\log n)$ bits of communication which are exchanged using $r_c = \Theta(1)$ symbols from the large alphabet. For an alphabet of constant size this is not possible and $r_c = \Theta(\log n)$ rounds of verification would be needed which is not possible. In Algorithm 1 the logarithmic amount of communication is used thrice: for the hash function seed, the

hash function value, and to coordinate the backtracking by communicating the transcript length.

1) Logarithmic Length Information to Coordinate Backtracking: The simplest idea for backtracking would be to have both parties go back some number of steps whenever a non-matching transcript is detected. This works well if both parties have equally long transcripts. Unfortunately, transcripts of different length are unavoidable because the adversary can easily make only one party backtrack while the other party continues. Then, if both parties always backtrack the same number of steps, both parties might reverse correct parts of the transcript without getting closer to each other. This means that with transcripts of different length the parties need to first and foremost come to realize which party is ahead and thus has to backtrack to the transcript length of the other party. Unfortunately, an adversarial channel can easily create transcript length differences of $\Theta(n\epsilon)$ steps between the two parties. For this it completely interrupts the communication of a simulation, as an "attacker in the middle", at a given point of time and simulates a faulty party with Alice, making her backtrack, while simulating a fully compliant party with Bob, making him go forward in an arbitrary wrong direction. With such large length differences it seems hard to achieve synchronization without sending logarithmic size length information.

Another problem is that, especially when dealing with adversarial channels, performing large re-synchronization steps is dangerous. In particular, if there is a way to make a party backtrack for a super-constant number of iterations triggered by only a constant amount of communication then the adversary can exploit this mechanism by faking this trigger. This would lead to $\omega(1)$ backtracking steps for every constant number of errors invested by the adversary and therefore make an optimal communication rate impossible.

2) Logarithmic Length Seeds and Hash Values: In the implementation of Algorithm 1 the seeds used to initialize the hash functions as well as the generated hash value itself are $\Theta(\log n)$ bits long. Looking at Lemma IV.1 reveals that this requirement comes both from the desire to make the collision probability small but also from the fact that we are hashing whole transcripts which are $O(n)$ bits long. One way to try to get around the later problem is to try hashing only the last few rounds. However, in the adversarial setting, it is unavoidable to have errors that go undetected for $n\epsilon$ rounds since the adversary can completely take over the conversation for this long. Another option would be to look for hash functions with a sub-logarithmic dependence on the length of the strings to be hashed. Unfortunately, this is not possible (see [12]).

B. Our Solutions

In this section we explain our solutions to the above problems and introduce the working parts and rationale behind Algorithm 3 and Algorithm 4.

1) Meeting Point Based Backtracking: We first explain how our algorithms coordinate their backtracking actions while exchanging only $O(1)$ bits per verification phase. In particular, we explain how the parties in our coding scheme determine where to and when to backtrack once they are aware that their transcripts are not in agreement or not synchronized.

Recall the second observation from Section V-A1 that parties cannot backtrack for more than a constant number of steps for every verification step, which consists of $O(1)$ bits of communication. In order to achieve this it is clear that parties might not be able to backtrack at all, even if non-matching transcripts were detected. Our algorithms implement this by maintaining at each party a threshold $\mathtt{k}$ for how far the party is willing to backtrack. For every iteration with an unresolved transcript inconsistency, that is, for every iteration since the last computation or backtracking step, this threshold increases by one without the party actually performing a backtracking step. The $\mathtt{k}$ values are kept synchronized between the two parties by including hashes of them in the verification phase and resetting them if too many discrepancies, measured by the error variable $\mathtt{E}$, are observed.

Now, with both parties having the same threshold $\mathtt{k}$ for how far they are able and willing to backtrack we define *meeting points* at which the parties can meet without having to communicate their position, i.e., their $\Theta(\log n)$ bit transcript length description. For this we create a *scale* $\tilde{\mathtt{k}}$ by rounding $\mathtt{k}$ to the next power of two, that is, $\tilde{\mathtt{k}} = 2^{\lfloor \log_2 \mathtt{k} \rfloor}$, and define the meeting points on this scale to be all multiples of $\tilde{\mathtt{k}}r$. A party is willing to backtrack to either of the two closest such meeting points, namely, $\mathtt{MP1} = \tilde{\mathtt{k}}r \left\lfloor \frac{|\mathtt{T}|}{\tilde{\mathtt{k}}r} \right\rfloor$ and $\mathtt{MP2} = \mathtt{MP1} - \tilde{\mathtt{k}}r$. It is easy to see that these meeting points are consistent and have the property that any two parties with the same scale $\tilde{\mathtt{k}}$ and a difference of $l^- < 2\tilde{\mathtt{k}}$ have at least one common meeting point up to which their transcripts agree. In each verification phase both parties send hash values of their transcripts up to these two meeting points, in the hope to find a match. We note that for a scale $\tilde{\mathtt{k}}$ there are $0.5\tilde{\mathtt{k}}$ hash comparisons generated during the time both parties look for a common meeting point at this scale $\tilde{\mathtt{k}}$. If most of these hashes, e.g., $0.4\tilde{\mathtt{k}}$ many, indicate a match a party backtracks to this point.

A potential function argument very similar to the one given for Algorithm 1 in Section IV-C, except for obviously involving many more cases, shows that this backtracking synchronization works as well as before while communicating only small hashes instead of logarithmic bit-sized length information.

2) Hash Values and Seeds: Next, we explain the strategies we use to reduce the communication overhead in the verification phase stemming from large hash values and seeds. The discussion regarding the seed length can be found in [12]. The reader is however highly encouraged to read it before trying to understand the hashing analysis.

Constant Size Hash Values: We concentrate on reducing the size of the hash values to a constant. What comes to the rescue here is the observation that hashing only makes one-sided errors, that is, it only confuses different strings for equal but never the other way around since hash values of the same string will always match. Since the primary cause for a non-matching transcript is an iteration with an error one would

furthermore expect that there are few, say $O(n\epsilon)$, opportunities for such a hash collision to happen. This would make it possible to have a constant hash collision probability without increasing the number of hash collisions beyond $\Theta(n\epsilon)$. This intuition is indeed correct and can be formalized relatively easily, as shown in [12].

VI. Our Coding Schemes

A. The Robust Randomness Exchange Subroutine

The Robust Randomness Exchange Subroutine is used to exchange some randomness at the beginning of the algorithm using an error correcting code which is then stretched to a longer δ=biased pseudo random string of length l using [14]. This string is then used by both parties to provide the random seeds for selecting the hash functions in each iteration (see also [12]).

Algorithm 2 Robust Randomness Exchange(l,δ)

Input: desired number of bits l and bias δ of the shared randomness
Output: shared random string R of length l and bias δ
$l' = \Theta(\log \delta + \log l)$
$C \leftarrow$ ECC $\{0,1\}^{l'} \rightarrow \{0,1\}^{\Theta(l'+nH(\epsilon))}$ with distance $4n\epsilon$
if Alice **then**
    $R' \leftarrow$ uniform random bit string of length l'
    Transmit $C(R')$ to Bob
else if Bob **then**
    Receive C' from Alice
    $R' \leftarrow$ Decoding of C'
$R \leftarrow \delta$-biased pseudo random string of length l derived from R'

B. The Inner Product Hash Function

In our algorithms we use the following, extremely simple, *inner product hash function*, which allows for an easy analysis given the δ-biased property of the shared random seed:

Definition VI.1 (Inner Product Hash Function)**.** *For any input length L and any output length o we define the inner product hash function $h_S(.)$ as doing the following: For a given binary seed S of length at least $2oL$ it takes any binary input string X of length $l \leq L$, concatenates this input with its length $\tilde{X} = (S, |S|)$ to form a string of length $\tilde{l} = |\tilde{X}| = |X| + \lceil \log_2 |X| \rceil \leq 2L$ and then outputs the o inner products $\left\langle \tilde{X}, S[i \cdot 2L + 1, i \cdot 2L + \tilde{l}] \right\rangle$ for every $i \in [0, o-1]$.*

The next corollary states the trivial fact that the inner product hash function is a reasonable hash function with collision probability exponential in its output length if a (huge) uniformly random seed is used. It also states that replacing this uniform seed by a δ-biased one does not change the outcome much. This follows directly from the definition of δ-bias:

Corollary VI.2. *Consider a pairs of binary strings $X \neq Y$ each of length at most L, and suppose h is the inner product hash function for input length L and any output length o. Suppose furthermore that S is seed string of length at least $n \cdot 2oL$ which is sampled independently of X, Y. The collision probability $P[h_S(X) = h_S(Y)]$ is exactly 2^{-o} if S is sampled from the uniform distribution. Furthermore, if the seed S is sampled from a δ-biased distribution the collision probability remains at most $2^{-o} + \delta$.*

Lastly, the next lemma summarizes the advantage of the inner product hash function in combination with a δ-biased seed, namely that this bias translates directly to the exact same bias on the output distribution. This uses the above mentioned fact from [14] that δ-bias also extends beyond variables to any set of linearly independent tests:

Lemma VI.3. *Consider n pairs of binary strings $(X_1, Y_1), \ldots, (X_n, Y_n)$ where each string is of length at most L, and suppose h is the inner product hash function for input length L and any output length o. Suppose furthermore that S is a random seed string of length at least $n \cdot 2oL$ which is sampled independently of the X and Y inputs and is cut into n strings $S_1, S_2, \ldots, S_n$. Then the output distribution $(x_1, \ldots, x_n) = (h_{S_1}(X_2) - h_{S_2}(Y_1), \ldots, h_{S_n}(X_2) - h_{S_n}(Y_1))$ for a S sampled from a δ-biased distribution is δ-statistically close to the output distribution for a uniformly sampled S for which each x_i is equal to zero if $X_i = Y_i$ and independently uniformly random otherwise (which also implies $P[x_i = 0] = 2^{-o}$).*

C. Our Coding Schemes

Algorithm 3 Coding Scheme for Oblivious Adv. Channels

$\Pi \leftarrow n$-round protocol to be simulated + final confirmation steps
$hash \leftarrow$ inner product hash fam. with $o = \Theta(1)$ and $s = \Theta(n)$
$r_c \leftarrow \Theta(1)$; $r \leftarrow \left\lceil \sqrt{\frac{r_c}{\epsilon}} \right\rceil$; $R_{total} \leftarrow \lceil n/r + 65n\epsilon \rceil$; $\mathtt{T} \leftarrow \emptyset$
Reset Status: $\mathtt{k}, \mathtt{E}, \mathtt{v1}, \mathtt{v2} \leftarrow 0$
R = Robust Randomness Exchange($l = R_{total} \cdot s$, $\delta = 2^{-\Theta(\frac{n}{r}o)}$)
for R_{total} iterations **do**
    $\mathtt{k} \leftarrow \mathtt{k} + 1$; $\tilde{\mathtt{k}} \leftarrow 2^{\lfloor \log_2 \mathtt{k} \rfloor}$
    $\mathtt{MP1} \leftarrow \tilde{\mathtt{k}} r \left\lfloor \frac{|\mathtt{T}|}{\tilde{\mathtt{k}} r} \right\rfloor$; $\mathtt{MP2} \leftarrow \mathtt{MP1} - \tilde{\mathtt{k}} r$ ▷ **Verification Phase**
    $\mathtt{S} \leftarrow s$ new preshared random bits from R
    Send $(hash_{\mathtt{S}}(\mathtt{k}), hash_{\mathtt{S}}(\mathtt{T}), hash_{\mathtt{S}}(\mathtt{T}[1, \mathtt{MP1}]), hash_{\mathtt{S}}(\mathtt{T}[1, \mathtt{MP2}]))$
    Receive $(H'_{\mathtt{k}}, H'_{\mathtt{T}}, H'_{\mathtt{MP1}}, H'_{\mathtt{MP2}})$;
    $h(.) = hash_{\mathtt{S}}(.)$
    $(H_{\mathtt{k}}, H_{\mathtt{T}}, H_{\mathtt{MP1}}, H_{\mathtt{MP2}}) \leftarrow (h(\mathtt{k}), h(\mathtt{T}), h(\mathtt{T}[1, \mathtt{MP1}]), h(\mathtt{T}[1, \mathtt{MP2}]))$
    if $H_{\mathtt{k}} \neq H'_{\mathtt{k}}$ **then**
        $\mathtt{E} \leftarrow \mathtt{E} + 1$
    else
        if $H_{\mathtt{MP1}} \in \{H'_{\mathtt{MP1}}, H'_{\mathtt{MP2}}\}$ **then**
            $\mathtt{v1} \leftarrow \mathtt{v1} + 1$
        else if $H_{\mathtt{MP2}} \in \{H'_{\mathtt{MP1}}, H'_{\mathtt{MP2}}\}$ **then**
            $\mathtt{v2} \leftarrow \mathtt{v2} + 1$
    if $\mathtt{k} = 1$ and $H_{\mathtt{T}} = H'_{\mathtt{T}}$ and $\mathtt{E} = 0$ **then** ▷ **Computation Phase**
        continue computation and transcript $\mathtt{T}$ for r steps
        Reset Status: $\mathtt{k}, \mathtt{E}, \mathtt{v1}, \mathtt{v2} \leftarrow 0$
    else
        do r dummy communications
    if $2\mathtt{E} \geq \mathtt{k}$ **then** ▷ **Transition Phase**
        Reset Status: $\mathtt{k}, \mathtt{E}, \mathtt{v1}, \mathtt{v2} \leftarrow 0$
    else if $\mathtt{k} = \tilde{\mathtt{k}}$ and $\mathtt{v1} \geq 0.4 \cdot \tilde{\mathtt{k}}$ **then**
        rollback computation and transcript $\mathtt{T}$ to position $\mathtt{MP1}$
        Reset Status: $\mathtt{k}, \mathtt{E}, \mathtt{v1}, \mathtt{v2} \leftarrow 0$
    else if $\mathtt{k} = \tilde{\mathtt{k}}$ and $\mathtt{v2} \geq 0.4 \cdot \tilde{\mathtt{k}}$ **then**
        rollback computation and transcript $\mathtt{T}$ to position $\mathtt{MP2}$
        Reset Status: $\mathtt{k}, \mathtt{E}, \mathtt{v1}, \mathtt{v2} \leftarrow 0$
    else if $\mathtt{k} = \tilde{\mathtt{k}}$ **then**
        $\mathtt{v1}, \mathtt{v2} \leftarrow 0$
Output the outcome of Π corresponding to transcript $\mathtt{T}$

Algorithm 4 Coding Scheme for Fully Adversarial Channels

1: $\Pi \leftarrow n$-round protocol to be simulated + final confirmation steps
2: $hash_1 \leftarrow$ inner product hash fam. with $o_1 = \Theta(\log \frac{1}{\epsilon})$ and $s_1 = \Theta(n)$
3: $hash_2 \leftarrow$ hash fam. with $p_2 = 0.1$, $o_2 = \Theta(1)$, and $s_2 = \Theta(\log\log \frac{1}{\epsilon})$
4: $R_{total} \leftarrow \lceil n/r \rceil + \Theta(n\epsilon)$; $r_c \leftarrow \Theta(\log\log \frac{1}{\epsilon})$; $r \leftarrow \left\lceil \sqrt{\frac{r_c}{\epsilon}} \right\rceil$; $\mathtt{T} \leftarrow \emptyset$
5: Reset Status: $\mathtt{k}, \mathtt{E}, \mathtt{v1}, \mathtt{v2} \leftarrow 0$
6: R = Robust Randomness Exchange($l = R_{total} \cdot s_1$, $\delta = 2^{-\Theta(\frac{n}{r})}$)
7: **for** R_{total} iterations **do**
8: $\quad \mathtt{k} \leftarrow \mathtt{k} + 1$; $\tilde{\mathtt{k}} \leftarrow 2^{\lfloor \log_2 \mathtt{k} \rfloor}$
9: $\quad \mathtt{MP1} \leftarrow \tilde{\mathtt{k}} r \left\lfloor \frac{|\mathtt{T}|}{\tilde{\mathtt{k}} r} \right\rfloor$; $\mathtt{MP2} \leftarrow \mathtt{MP1} - \tilde{\mathtt{k}} r$ ▷ **Verification Phase**
10: $\quad \mathtt{S}_1 \leftarrow s_1$ new preshared random bits from R
11: $\quad \mathtt{S}_2 \leftarrow s_2$ "fresh" random bits
12: $\quad hash(.) = hash_{2,\mathtt{S}_2}(hash_{1,\mathtt{S}_1}(.))$
13: $\quad$ Send $(\mathtt{S}_2, hash(\mathtt{k}), hash(\mathtt{T}), hash(\mathtt{T}[1,\mathtt{MP1}]), hash(\mathtt{T}[1,\mathtt{MP2}]))$
14: $\quad$ Receive $(\mathtt{S}'_2, H'_\mathtt{k}, H'_\mathtt{T}, H'_\mathtt{MP1}, H'_\mathtt{MP2})$;
15: $\quad h'(.) = hash_{2,\mathtt{S}'_2}(hash_{1,\mathtt{S}_1}(.))$
16: $\quad (H_\mathtt{k}, H_\mathtt{T}, H_\mathtt{MP1}, H_\mathtt{MP2}) \leftarrow (h'(\mathtt{k}), h'(\mathtt{T}), h'(\mathtt{T}[1,\mathtt{MP1}]), h'(\mathtt{T}[1,\mathtt{MP2}]))$
17: $\quad$ **Remaining Code as in Lines 14 to 36 in Algorithm 3**

VII. Analyses and Proofs of Correctness

Theorem VII.1. *Suppose any n-round protocol Π using any alphabet Σ. Algorithm 3 is a computationally efficient randomized coding scheme which given Π, with probability $1 - 2^{-\Theta(n\epsilon)}$, robustly simulates it over any oblivious adversarial error channel with alphabet Σ and error rate ϵ. The simulation uses $n(1 + \Theta(\sqrt{\epsilon}))$ rounds and therefore achieves a communication rate of $1 - \Theta(\sqrt{\epsilon})$.*

Theorem VII.2. *Suppose any n-round protocol Π using any alphabet Σ. Algorithm 4 is a computationally efficient randomized coding scheme which given Π, with probability $1 - 2^{-\Theta(n\epsilon)}$, robustly simulates it over any fully adversarial error channel with alphabet Σ and error rate ϵ. The simulation uses $n(1 + \Theta(\sqrt{\epsilon \log\log \frac{1}{\epsilon}})$ rounds and therefore achieves a communication rate of $1 - \Theta(\sqrt{\epsilon \log\log \frac{1}{\epsilon}})$.*

Proof Outline: We use the same proof structure as already introduced in Section IV-C. In particular, we show that the algorithm terminates correctly by defining a potential Φ. We prove that any iteration without an error or hash collision increases the potential by at least a constant while any iteration with an error or hash collision reduces the potential at most by some constant. We do this in two steps: First we show that this statement is true for the computation and verification phase of each iteration only. We then show that any transition in the transition phase does not decrease the potential. As a last step, we bound the number of hash collisions to be of the same order as the number of errors. This is the sole part in which the analyses of Algorithm 3 and Algorithm 4 differ.

Potential: The potential Φ is based on the variables $\mathtt{k}$, $\mathtt{E}$, and $\mathtt{T}$ of both parties. For these variables we use a subscript A or B to denote the value of the variable for Alice and Bob respectively. We also denote with the subscript AB the sum of both these variables, e.g., $k_{AB} = k_A + k_B$. To define Φ we need the following intermediate quantities:

As before, we define the amount of agreement l^+ and disagreement l^- between the two paths computed at Alice and Bob as $l^+ = \left\lfloor \frac{1}{r} \max\{l' \text{ s.th. } \mathtt{T}_A[1,l'] = \mathtt{T}_B[1,l']\} \right\rfloor$ and $l^- = \frac{|\mathtt{T}_A| + |\mathtt{T}_B|}{r} - 2l^+$.

For sake of the analysis we also define two variables BVC_A and BVC_B which count the contribution of hash collisions and corruptions to $\mathtt{v1}$ and $\mathtt{v2}$ at Alice and Bob. In any iteration in which $\mathtt{v1}$ of either party increases in Line 18 without $\mathtt{T}[1,\mathtt{MP1}]$ matching either $\mathtt{T}[1,\mathtt{MP1}]$ or $\mathtt{T}[1,\mathtt{MP2}]$ of the other party we count this as a bad vote and increase *both* BVC_A and BVC_B. Similarly, we increase both BVC values if $\mathtt{v2}$ of a party increases in Line 20 without $\mathtt{T}[1,\mathtt{MP2}]$ matching either $\mathtt{T}[1,\mathtt{MP1}]$ or $\mathtt{T}[1,\mathtt{MP2}]$ of the other party. On the other hand, if one such match occurs but the corresponding vote does not increase, e.g., due to a corruption, then we call this an uncounted vote and also increase BVC_A and BVC_B by one. With every status reset (Lines 27, 30 and 33) we also set the BVC count of this party to be zero. We remark that the BVC values are not known to either party; they are merely used to facilitate our analysis.

To weight the various contributions to the potential we use the constants $1 < C_2 < C_3 < C_4 < C_5 < C_6$, which are chosen such that C_i is sufficiently large depending only on C_j with $j < i$. The potential Φ is now defined to be

$l^+ - C_3 \cdot l^- + C_2 \cdot \mathtt{k}_{AB} - C_5 \cdot \mathtt{E}_{AB} - 2C_6 \cdot BVC_{AB}$ if $\mathtt{k}_A = \mathtt{k}_B$ and

$l^+ - C_3 \cdot l^- - 0.9C_4 \cdot \mathtt{k}_{AB} + C_4 \cdot \mathtt{E}_{AB} - C_6 \cdot BVC_{AB}$ otherwise.

Lemma VII.3. *In every computation and verification phase the potential decreases at most by a fixed constant, regardless of the number of errors and hash collisions. Furthermore, in the absence of an error or hash collision the potential strictly increases by a at least one.*

Proof. All quantities on which the potential depends change at most by a constant during any computation and verification phase. The maximum potential change is therefore at most a constant. Now we consider the case that no error or hash collision happened. In this case, the BVC_{AB} value does not change. Furthermore, computation only happens if $\mathtt{T}_A = \mathtt{T}_B$ which implies that the $l^+ - C_3 \cdot l^-$ part of the potential does not decrease. Lastly, both $\mathtt{k}_A$ and $\mathtt{k}_B$ increase by one and if they are not equal $\mathtt{E}_A$ and $\mathtt{E}_B$ increase by one, too. In the first case the increase of $\mathtt{k}_{AB}$ leads to a total potential increase of $2C_2 > 1$ in the later case the increase of $\mathtt{k}_{AB}$ and $\mathtt{E}_{AB}$ leads to a total potential change of $2(-0.9C_4 + C_4)$ which is at least one for sufficiently large C_4. The potential therefore strictly increases by at least one in the computation and update phase when no error or hash collision happens. □

Lemma VII.4. *In every iteration the potential decreases at most by a fixed constant, regardless of the number of errors and hash collisions. Furthermore, in the absence of an error or hash collision the potential strictly increases by at least one.*

Next, we show that our hash function families and the randomness used in our algorithms is strong enough to ensure that the total number of hash collisions is small, namely

comparable to the number of errors. This allows us to treat any iteration with a hash collisions as adversarially corrupted and thus equivalent to an iteration with errors.

We start by showing that the potential Φ cannot grow too fast. In particular, it grows naturally by one per iteration when a correct computation step is performed. On the other hand, any corruption cannot increase this by more than a constant per error:

Lemma VII.5. *The total potential Φ after R iterations is at most $R + 20C_2n\epsilon$.*

Now we can show the number of hash collisions in Algorithm 3 to be small:

Lemma VII.6. *For any protocol Π and any oblivious adversary the number of iterations suffering a hash collision in Algorithm 3 is at most $\Theta(\epsilon n)$, with probability $1 - 2^{-\Theta(n\epsilon)}$.*

Next, we show that the $hash_1$ hash function in Algorithm 4 also causes at most $\Theta(\epsilon n)$ hash collisions, even for a fully adversarial channel:

Lemma VII.7. *For any protocol Π and any fully adversarial channel the number of iterations with hash collisions due to first hashing with the hash function $hash_1$ in Algorithm 4 is at most $\Theta(\epsilon n)$, with probability $1 - \epsilon^{\Theta(n\epsilon)}$.*

Lemma VII.7 implies that even if we treat $hash_1$ hash collisions in Algorithm 4 as errors then this only increases the number of possible errors by a constant factor. We can therefore restrict ourselves in the next lemma to analyzing hash collisions due to the $hash_2$ hash function. This hash function however only needs to map the short $o_1 = \Theta(\log \frac{1}{\epsilon})$ long hash values to a constant output of length $o_2 = \Theta(1)$. Using the hash functions from Lemma IV.1 for this only $\Theta(\log\log \frac{1}{\epsilon})$ bit sized seeds are necessary. Similar to Corollary IV.5 these small seeds are sampled afresh in every iteration which makes the hash collisions due to $hash_2$ being dominated by independent Bernoulli($\Theta(1)$) trials. Again, following the arguments in Lemma VII.6 having more than $\Theta(n\epsilon)$ such hash collisions has a probability of at most $2^{-\Theta(n\epsilon)}$:

Corollary VII.8. *For any protocol Π and any fully adversarial channel the number of iterations with hash collisions due to hashing with the hash function $hash_2$ in Algorithm 4 is at most $\Theta(\epsilon n)$, with probability $1 - 2^{-\Theta(n\epsilon)}$.*

With these $\Theta(n\epsilon)$ bounds on the total number of hash collisions in both Algorithm 3 and Algorithm 4 we can prove our main results:

Proof of Theorem VII.1 and Theorem VII.2. Lemmas VII.6 and VII.7 and Corollary VII.8 show that both in Algorithm 3 and Algorithm 4 at most $\Theta(n\epsilon)$ hash collisions or errors happen. Lemma VII.4 shows that the potential drop in these rounds is bounded by a fixed $\Theta(n\epsilon)$ while the potential increases by at least one in the remaining $R_{total} - \Theta(n\epsilon)$ rounds. For a sufficiently large $R_{total} = \lceil n/r \rceil + \Theta(n\epsilon)$ this implies a potential of at least $\Phi > \lceil n/r \rceil + \Theta(n\epsilon)$ in the end. Following the arguments in Lemma VII.5 we get that $l^+ \geq \lceil n/r \rceil$ which implies that the parties agree upon the first n symbols of the execution of Π and therefore both output the correct outcome.

The total round complexity of the main loop in both algorithms is $R_{total}(r + r_c) = (\lceil n/r \rceil + \Theta(n\epsilon))r(1 + \frac{r_c}{r}) = n(1 + \Theta(r\epsilon))(1 + \frac{r_c}{r}) = n(1 + \Theta(r\epsilon + \frac{r_c}{r}))$ and in both algorithms r is set to the (asymptotically) optimal value $r = \lceil \sqrt{\frac{r_c}{\epsilon}} \rceil$ which makes this round complexity equal to $n(1 + \Theta(\sqrt{r_c\epsilon}))$. In Algorithm 3 $r_c = \Theta(1)$ which leads to a round complexity of $n(1 + \Theta(\sqrt{\epsilon}))$ in the main loop. In Algorithm 4 $r_c = \Theta(\log\log \frac{1}{\epsilon})$ which leads to a round complexity of $n(1+\Theta(\sqrt{\epsilon \log\log \frac{1}{\epsilon}}))$ in the main loop. In both algorithms the communication performed by the randomness exchange is $\Theta(n\sqrt{\epsilon})$ many rounds and therefore negligible. This shows the communication rate of Algorithm 3 to be $1 - \Theta(\sqrt{\epsilon})$ and the communication rate of Algorithm 4 to be $1 - \Theta(\sqrt{\epsilon \log\log \frac{1}{\epsilon}})$ as desired. □

REFERENCES

[1] Z. Brakerski and Y. Kalai. Efficient interactive coding against adversarial noise. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 160–166, 2012.

[2] Z. Brakerski and M. Naor. Fast algorithms for interactive coding. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 443–456, 2013.

[3] M. Braverman. Coding for interactive computation: progress and challenges. In *Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1914–1921, 2012.

[4] M. Braverman and K. 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)*, 2014.

[5] M. Braverman and A. Rao. Towards coding for maximum errors in interactive communication. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 159–166, 2011.

[6] K. Efremenko, R. Gelles, and B. Haeupler. Maximal noise in interactive communication over erasure channels and channels with feedback. In *arXiv*, 2014.

[7] M. Franklin, R. Gelles, R. Ostrovsky, and L. J. Schulman. Optimal coding for streaming authentication and interactive communication. In *Proceedings of International Cryptology Conference (CRYPTO)*, pages 258–276, 2013.

[8] R. Gelles and B. Haeupler. Capacity of interactive communication over erasure channels and channels with feedback. In *arXiv*, 2014.

[9] R. Gelles, A. Moitra, and A. Sahai. Efficient and explicit coding for interactive communication. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 768–777, 2011.

[10] M. Ghaffari and B. Haeupler. Optimal Error Rates for Interactive Coding II: Efficiency and List decoding. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 2014.

[11] M. Ghaffari, B. Haeupler, and M. Sudan. Optimal Error Rates for Interactive Coding I: Adaptivity and other settings. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 794–803, 2014.

[12] B. Haeupler. Interactive channel capacity revisited. In *arXiv*, 2014.

[13] G. Kol and R. Raz. Interactive channel capacity. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 715–724, 2013.

[14] J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing (SICOMP)*, 22(4):838–856, 1993.

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