# A Constant-Rate Compiler for MPC over Noisy Networks

Ran Gelles[1], Carmit Hazay[1], Manuj Mukherjee[2], Jaspal Singh[3,4], Arun Yeragudipati[3], and Vassilis Zikas[4]

[1] Bar-Ilan University
[2] Indraprastha Institute of Information Technology, Delhi
[3] Purdue University
[4] Georgia Institute of Technology
{Carmit.Hazay,ran.gelles}@biu.ac.il, manuj@iiitd.ac.in, sing1361@purdue.edu, ysarun1999@gmail.com, vzikas@gatech.edu,

**Abstract.** The study of efficient multi-party computation (MPC) has been a central focus of cryptographic research, leading to a broad range of techniques across diverse practical models. However, the vast majority of this work assumes reliable communication channels and largely ignores network-level noise, a fundamental characteristic of modern communication systems.

In this work we study information-theoretic MPC over *noisy* networks in the multi-party setting. We construct an $n$-party MPC protocol that is secure against semi-honest adversaries corrupting a majority of the parties assuming all communication is carried over BSC channels with a fixed bit-flip probability. Our construction incurs only constant-factor communication overhead compared to the underlying MPC protocol over noiseless channels: specifically, it computes any circuit $C$ with communication complexity $O(n^2(|C|+\kappa))$, where $\kappa$ is a statistical security parameter. This substantially improves the naïve approach, which can only guarantee super-linear communication, namely, $O(n^2|C|(\kappa+\log|C|))$. To achieve this result, we introduce a novel multi-party *interactive coding* scheme tailored to cryptographic protocols that rely on correlated randomness. We show that combining this coding scheme with the classical GMW protocol yields a noise-resilient MPC protocol that preserves its security guarantees while incurring only linear overhead.

A central technical challenge we address is a subtle correctness issue that arises in the simulation of protocols with correlated randomness: without careful synchronization, different parties may use inconsistent correlated randomness for what is logically the same protocol step, causing the computation to fail even in the absence of noise. Our novel interactive coding scheme ensures that all parties use consistent correlated randomness in every simulated round of the MPC. Surprisingly, this issue also affects the two-party simulation over noisy channels [Gelles, Paskin-Cherniavsky, and Zikas , SCN '18], leaving an unaddressed gap in its analysis. Our techniques provide a way to resolve this gap.

## 1 Introduction

Secure Multi-Party Computation (MPC) enables a set of $n$ parties, each holding a private input $x_1,\ldots,x_n$, to jointly compute a public function $f(x_1,\ldots,x_n)$ while guaranteeing that any subset of corrupted parties learns nothing beyond the prescribed output. Since the seminal work of Yao [Yao82], MPC has become a central theme in cryptography, with extensive research exploring diverse adversarial models, computational assumptions, and efficiency goals. Despite this rich literature, relatively little attention has been given to the feasibility of MPC over *noisy* communication channels, that is, when random interference, hardware faults, or adversarial manipulation may corrupt messages sent by honest parties. Such noise is inherent in many real-world networks, especially wireless and certain Internet-based systems, where reliable channels cannot be assumed.

One might be tempted to think that specialized techniques for noisy channels are unnecessary. If a protocol $\pi$ is resilient to malicious node corruptions over reliable channels, then perhaps it should also remain secure over simple noisy channels such as $\mathsf{BSC}_\varepsilon$. After all, channel noise may seem analogous

to adversarial deviations in the noiseless model. This intuition, however, is incorrect. Security against malicious behavior guarantees privacy even when a party deviates from the protocol arbitrarily, but it does not ensure the correctness of the corrupted parties' outputs. In contrast, in security over noisy channels, the protocol must guarantee that all parties obtain the correct output despite channel errors, even if no party deviates from the prescribed algorithm (as in the semi-honest setting). Consequently, a maliciously secure protocol over reliable channels *does not* imply a semi-honest secure protocol over noisy channels; bridging this gap requires mechanisms that explicitly cope with transmission errors while preserving privacy, even though errors may accumulate and interact across many rounds in ways that are hard to localize and may only be detected much later in the execution.[1]

Motivated by this gap, we study MPC over noisy channels in the *multi-party* setting. We focus on semi-honest adversaries corrupting up to $n-1$ parties and on the widely studied model of random noise instantiated with the *binary symmetric channel* $\mathsf{BSC}_\varepsilon$, which flips each transmitted bit independently with fixed (constant) probability $\varepsilon \in (0, 1/2)$.

A standard method for coping with such noise is to apply an error-correcting code (ECC) (see e.g., [Rot06]) to every transmitted message. With sufficiently long encodings, a receiver can correctly decode any message with high probability despite random corruptions. Consequently, a natural approach to obtaining noise-resilient MPC protocols is to encode each message of an MPC protocol $\pi$ using an ECC, so that when these encodings are transmitted over $\mathsf{BSC}_\varepsilon$ they can be correctly decoded, yielding for each recipient a view identical to that of $\pi$. While this approach yields a secure, noise-resilient MPC protocol, it suffers from two major drawbacks.

1. *Super-Linear Communication Blowup.* Encoding a single message with an ECC (e.g., as in Lemma 1) incurs linear communication blowup while achieving an exponentially small decoding-failure probability in the codeword length. However, this failure probability applies per message. Therefore, by a union bound, the probability that *some* message in an ECC-encoded execution of a protocol $\pi$ is corrupted increases by a factor proportional to the number of messages in $\pi$, and must be compensated for by increasing the encoding length.
   Let us give a concrete example. Let $\pi'$ denote the ECC-encoded version of an MPC protocol $\pi$, and let $\kappa$ be the statistical security parameter, i.e., we require the overall failure probability to be at most $2^{-\kappa}$. To achieve this guarantee, the ECC must use an encoding length of at least $\Omega(\kappa + \log \mathsf{msg}(\pi))$, where $\mathsf{msg}(\pi)$ denotes the number of messages communicated in $\pi$.[2]
   Many standard MPC protocols, e.g., [GMW87, BGW88, CCD88, DN03] have message complexity $O(n^2|C|)$. Consequently, the ECC-encoded protocol incurs total communication of
   $$O\left(\mathsf{msg}(\pi)(\kappa + \log \mathsf{msg}(\pi))\right) = O\left(n^2|C|(\kappa + \log |C|)\right),$$
   which represents a super-linear overhead in the circuit size compared to the noiseless execution.
2. *Vulnerability to Stronger Noise.* Independent per-message coding offers no protection against structured or adversarial noise. An adversary that can corrupt even a small fraction of bits can concentrate its budget on a single codeword and completely invalidate it. Although our focus in this paper is on random noise, we view our contribution as a stepping stone toward MPC in stronger settings. These considerations suggest that a fundamentally different technique

[1] We set aside the fact that malicious security does not, in general, imply the weaker semi-honest security. The point here is to illustrate that resilience to node corruptions does not imply even a basic form of channel resilience against channel attacks.

[2] Here $\mathsf{msg}(\pi)$ counts the number of *distinct messages* exchanged in the protocol, since each such message must be separately encoded.

is required, one that exploits the interactive structure of the protocol rather than encoding messages independently, and that can capture and maintain the global dependencies created by rewinding, synchronization, and correlated randomness.

In light of the above drawbacks, we ask whether it is possible to obtain secure, *noise resilient* computation while incurring only linear overhead in communication with respect to the noiseless MPC. Importantly, the goal is not only to avoid the communication blowup caused by independent per-message error correction, but also to develop techniques that are compatible with as large class of MPC schemes, so that our solutions could naturally extend to stronger noise models in the future.

In this work, we are interested in developing information-theoretic MPC protocols that do not rely on an honest majority assumption and remain secure even when an arbitrary subset of parties may be semi-honestly corrupted.[3] This choice of model is especially natural in the noisy-channel setting, where the communication medium itself provides an information-theoretic cryptographic resource. Concretely, in this work we address the following problem:

> *Can general information-theoretic secure MPC be achieved over noisy* $\mathsf{BSC}_\varepsilon$ *channels, while tolerating a dishonest majority of semi-honest adversaries, and with asymptotic communication complexity that scales linearly with the circuit size? Concretely, is it possible to achieve communication complexity* $O_\varepsilon(n^2|C|)$ *for any circuit* $C$*?*[4]

For the two-party case, Gelles, Paskin-Cherniavsky, and Zikas [GPZ18] showed that replacing independent per-message error correction with interactive coding (IC) scheme [Sch92, Sch93, Gel17] enables information-theoretically secure computation over $\mathsf{BSC}_\varepsilon$ with linear communication overhead when one party is semi-honest. Interactive coding schemes compile a protocol $\pi$ designed for noiseless channels into a functionally equivalent protocol $\pi'$ that tolerates channel noise, such as $\mathsf{BSC}_\varepsilon$ and, in some works, even adversarial noise. Most known IC schemes are *constant-rate*, meaning that the communication of the compiled protocol grows linearly with that of the original, i.e., $\mathtt{CC}(\pi') = O(\mathtt{CC}(\pi))$. By leveraging the protocol's interactive structure rather than encoding messages independently, the IC paradigm can overcome the drawbacks of per-message error correction.

However, the two-party construction of [GPZ18] does not extend to the multi-party setting due to two fundamental issues. First, it relies on a two-party IC scheme that protects each communication channel in isolation, but lacks mechanisms for providing resilience at the network level. Second, and more importantly, it does not guarantee coordinated use of the correlated randomness employed by the parties, *even in the two-party case*, leaving a gap in its analysis. This shortcoming stems from a subtle issue that we identify and analyze in Section 1.1 below (see Issue II), which was not addressed in [GPZ18].

In this work, we give an affirmative answer to the above question in the dishonest-majority, semi-honest setting. Specifically, we design a circuit-based MPC protocol that achieves information-theoretic security over noisy $\mathsf{BSC}_\varepsilon$ channels without relying on any additional setup assumptions.

**Theorem 1 (Main theorem, Informal).** *For any circuit* $C : \{\{0,1\}^m\}^n \to \{0,1\}^{m'}$*, and any* $\varepsilon \in (0, 1/2)$*, there exists an* $n$*-party protocol that evaluates* $C(x_1, \ldots, x_n)$ *for any inputs* $x_i \in \{0,1\}^m$ *over* $\mathsf{BSC}_\varepsilon$ *channels that is statistically secure against semi-honest adversaries corrupting* $\leqslant n-1$ *parties with communication complexity* $O_\varepsilon(n^2(|C| + \kappa))$*, where* $\kappa$ *is a statistical security parameter.*

[3] This also captures secure two-party computation protocols where one of the parties may be semi-honestly corrupted.
[4] $O_\varepsilon(\cdot)$ denotes bounds whose implicit constants may depend on the constant $\varepsilon$.

Our theorem can be interpreted as a compiler that receives a circuit $C$, and outputs a secure MPC protocol that evaluates $C$, preserving privacy even when communicating over noisy channels. Towards achieving this goal, we build on the paradigm of [GPZ18], replacing their two-party IC scheme with a multi-party scheme. However, none of the existing multi-party IC schemes (e.g., [RS94, JKL15, LV15, HS16, CGH19, ABE+19, CLPS20, GKR21, GKR22]) satisfy the requirements needed for secure MPC. While these schemes ensure the correctness of the simulated transcript, they are not privacy-preserving and may leak information even against an honest majority. Consequently, such schemes cannot be directly composed with MPC protocols over noisy channels.

We therefore develop a *novel interactive coding scheme* that guarantees the correct execution of any protocol over noisy channels while preserving its asymptotic communication complexity and simultaneously providing the properties required to maintain privacy. Applying this multi-party IC scheme to the classical GMW protocol [GMW87], in which each oblivious transfer (OT) is instantiated via communication over a noisy $\mathsf{BSC}_\varepsilon$ channel (e.g., using [CK88]), yields a noise-resilient secure MPC protocol. However, this approach incurs a large overhead, as each OT instance is replaced by $O(\text{poly}(\kappa))$ communicated bits.

To eliminate this bottleneck, we first use the noisy-channel OT construction of [HIKN08] to generate a large batch of OT correlations at *linear* cost: producing $\ell$ correlations using only $O(\ell)$ bits of communication. Second, we run the GMW protocol using these precomputed correlations, while applying our multi-party IC scheme to protect the interactive execution from channel noise. This combination yields a multi-party MPC protocol that is statistically secure against a semi-honest dishonest majority over noisy channels, with a constant communication overhead.

### 1.1 Technical Overview

Our technical contribution consists of two complementary components. First, we develop a new multi-party interactive coding scheme, presented in Section 3, that is both communication- and privacy-preserving for circuit-based protocols with gate-by-gate evaluation. Second, we analyze the security of the GMW protocol when encoded via this scheme. Central to our contribution is the identification of the properties that an IC scheme must satisfy to guarantee privacy in addition to error correction. In particular, we analyze what fundamentally goes wrong when prior IC schemes are applied directly to GMW-based computation. Once these requirements are isolated, we develop new mechanisms to address them. In this part, we outline the challenges of designing an information-theoretically secure MPC protocol over noisy channels with linear overhead, and present the main ideas underlying our construction. We begin by examining a natural but flawed approach that does not achieve security in this setting; this attempt serves as a useful template for our final protocol and clarifies the properties required from our IC scheme.

**The challenges of combining existing IC with GMW.** Our starting point is the multi-party IC scheme by Alon et al. [ABE+19], which allows parties to *simulate*, with high probability, any protocol $\pi$ over fully connected networks where each channel is an independent $\mathsf{BSC}_\varepsilon$. In this context, simulating a protocol means that each party obtains all the messages it should have obtained in a (noiseless) execution of $\pi$; hence, the party can compute its correct output of $\pi$. We base our construction on this scheme since it is the only known IC scheme that incurs a constant communication overhead over a clique: it simulates any $\pi$ in $O_\varepsilon(|\pi|)$ rounds of communication and succeeds with probability $1 - 2^{-\Omega_\varepsilon(\sqrt{n}|\pi|)}$. A natural first attempt is to apply this IC scheme to the

classic $n$-party GMW protocol [GMW87], originally designed for noiseless channels, and execute it over $\mathsf{BSC}_\varepsilon$. However, this direct compilation fails to produce a secure MPC protocol: two fundamental issues arise which we describe next.

*Issue I: Privacy violation caused by rewinding.* The first issue, already identified in [GPZ18] for the two-party case (based on the IC scheme of [Hae14]), stems from a potential privacy violation caused by rewinding: Resilience to noise requires rewinding the simulated protocol to a point preceding the detected corruption; however, in cryptographic protocols, such rewinding can compromise the security guarantees of the underlying computation [GSW15]. To be more concrete, consider the IC scheme of [ABE+19], which simulates $\pi$ round by round and triggers a rewind whenever an inconsistency is detected due to channel noise. In that case, the parties roll back the simulation to an earlier round of $\pi$ that appears consistent from the perspective of all parties. Applying this mechanism directly to the GMW protocol, however, is problematic. To see the issue, recall that in GMW, the parties evaluate a sequence of NAND gates, each implemented via two calls to an OT protocol (see Section 4.1). Rewinding a single round in the middle of such a gate may violate security, particularly if OT invocations are repeated or reused without proper safeguards. (see concrete example in Appendix A).

A natural way to avoid this leakage is to permit rewinding only to the beginning of the current NAND gate and to re-simulate that gate using fresh randomness [GPZ18]; thus, schemes that work in a round-by-round fashion could not be directly used. An alternative to the above is to use the multi-party IC schemes of [GKR21, GKR22], which already operate in large chunks. However, these schemes do not meet our needs, as they incur superlinear communication overhead. Instead, we modify the scheme of [ABE+19] so that the simulation proceeds in atomic *chunks* of multiple rounds at a time. The challenge here is purely technical: working in chunks changes the simulation's mechanics and requires a new correctness proof. See more details on our specific construction below.

While chunk-based rewinding prevents the leakage described above, re-simulating an OT instance with fresh randomness introduces additional difficulties that prior IC schemes do not address, as discussed next.

*Issue II: Correlation consistency.* The second, and more critical and subtle issue, stems from the reliance of $\pi$ on correlated randomness. In the GMW protocol, the parties must consume *consistent* correlated randomness when executing each OT instance to obtain a valid computation [Bea91].

Prior IC schemes, and in particular [ABE+19], allow different parties to simulate the same round of $\pi$ at different times. This flexibility is benign (and even desirable!) for deterministic protocols, or whenever the parties' randomness is independent. In our setting, however, it becomes harmful: parties may end up using different random OT instances for what is logically the same OT invocation in GMW. As a result, even when all transmitted messages appear mutually consistent, and there is no channel noise, the underlying computation may be invalid, leading to an incorrect output.

Crucially, such inconsistencies are invisible to existing IC mechanisms. From each party's local perspective, every message it sends is consistent with its own transcript and randomness, and therefore, the IC scheme has no reason to trigger a rewind. The error arises not from corrupted communication but from a misalignment of correlated state, a phenomenon outside the model considered by prior IC schemes.

Upon identifying this issue, we observe that it is not specific to the multi-party setting; the same problem arises even in the two-party case. However, the construction of [GPZ18] does not address

this subtlety. Nevertheless, a correct two-party secure simulation can be obtained by incorporating the techniques developed here.

**A new multi-party IC scheme.** Our main contribution is a new multi-party IC scheme that is tailored to cryptographic protocols with correlated randomness. In particular, our scheme enforces chunk-level simulation and includes novel mechanisms for maintaining global synchrony of simulated rounds across parties. To simplify the presentation, we explain how to resolve each issue in isolation; combining these approaches to a single scheme is straightforward.

*An IC scheme solving the first issue: Detailed Overview.* The first coding scheme, presented in Section D, non-trivially extends the IC scheme of [ABE+19] (which itself builds on the scheme of [RS94]) so that the simulation proceeds in chunks of $c$ rounds, rather than in a round-by-round fashion. Recall that the scheme in [RS94, ABE+19] simulates $\pi$ round-by-round, where each round is encoded via a mechanism known as *tree code* [Sch93] (see Def. 4). This is an online encoding method that guarantees the correct decoding of an ever-increasing prefix of the encoded information. In particular, even if some rounds are corrupted (in either the first or the second part), the parties will be able to detect this corruption after a few rounds. In this case, the parties exchange a special "rewind" symbol (still encoded via the tree-code) to indicate the need for a rewind.

Extending this approach to work in chunks is not completely straightforward: due to interactions, a party cannot encode information across multiple rounds in a single tree code symbol; this information becomes available only after those rounds are over. In the scheme we design, each chunk's simulation consists of two parts: *simulation* and *consistency check*. In the first part, the parties simulate $c$ rounds of $\pi$, where each round is separately encoded using a standard error-correcting code (Specifically, the A2A scheme; see Section B.1). In the second part, the parties check whether the simulation so far seems consistent by exchanging redundancy information: the information of the *entire* last chunk encoded via a tree code. If all transmissions are consistent with the received redundancy information, the simulation advances to the next chunk. Otherwise, during the simulation part of the next chunk, the party ignores the first part and sends a "rewind" instruction during the second part, instructing the other parties to rewind the simulation by a single chunk.

*An IC scheme solving the second issue: Detailed Overview.* Next, we address the requirement that all nodes use the same correlated randomness when simulating a given chunk. Our approach modifies the "consistency check" component of the coding scheme so that, at every round, a party verifies the following two properties. (1) As in the schemes of [RS94, ABE+19], every sent message must be consistent with the information received so far. (2) Additionally, each chunk of $\pi$ must be simulated by all parties at *exactly* the same round of the simulation scheme. To verify property (2), each party records the round numbers in which chunks that made it to the current "final" transcript were simulated (i.e., chunks that were not rewound and removed from the transcript). The party then processes the tree-code symbols received from each neighbor and deduces, for each simulated chunk, the round number at which that neighbor simulated the chunk. If the simulation of all chunks appears consistent and synchronous, the party proceeds to simulate the next chunk; otherwise, it rewinds.

However, rewinding a single chunk whenever an inconsistency is detected may be catastrophic to the coding scheme's success. To illustrate this issue, consider a scenario in which, at the end of some simulation round (say, round $t$), all parties have correctly simulated up to chunk $i$. Now suppose that in the next round (round $t+1$), a communication error causes party $P_j$ to believe that an inconsistency has occurred, and it therefore rewinds one chunk to $i-1$. At the same time, party $P_k$

continues its simulation and advances to chunk $i+1$. In the following round (round $t+2$), both parties detect inconsistencies: party $P_k$ detects the rewinding of $P_j$ and consequently rewinds one chunk, as it must re-simulate chunk $i-1$. However, party $P_j$ has just simulated chunk $i-1$ in round $t+2$, whereas party $P_k$ simulated it earlier, resulting in a synchronization inconsistency. If this situation is not handled carefully and party $P_j$ rewinds one chunk as well, it may happen that parties $P_j$ and $P_k$ continue rewinding all the way back to chunk 0, thereby erasing all progress made so far.

Instead, we introduce a "hold" instruction. When a party $P_j$ detects that it needs to re-simulate a chunk and that some other party $P_k$ is currently ahead of $P_j$, party $P_j$ *does not rewind* but instead "holds" its position and waits until $P_k$ rewinds. Once all parties become synchronized again, they jointly re-simulate the next chunk at the same simulation round, thereby maintaining the required synchronization. Introducing a hold instruction complicates the parties' behavior and, in turn, the correctness analysis of the coding scheme. Fortunately, we show that the scheme converges correctly with high probability. Moreover, although the simulation may need to "pause" until all parties rewind to the same point, we prove that a single channel corruption delays the simulation by only $O(1)$ rounds, thereby guaranteeing linear overall blowup. The detailed analysis appears in Section 3.1.

To the best of our knowledge, this is the first multi-party interactive coding scheme whose analysis explicitly handles randomized protocols with evolving correlated randomness, where synchronization and error recovery must preserve both correctness and privacy.

**MPC over noisy channels.** The classic GMW protocol [GMW87] over noiseless channels for secure multi-party computation (to compute any circuit over the inputs) proceeds in three phases: (i) input sharing: each party secret-shares its input among all parties; (ii) circuit evaluation: the parties jointly evaluate the circuit gate by gate using local computation and correlated randomness; and (iii) output reconstruction: the parties reconstruct the final output of the circuit from their shares.

Our proposed MPC protocol over $\mathsf{BSC}_\varepsilon$ based on the classic GMW protocol and the proposed IC scheme, has the following outline:

1. In the input-sharing and output-reconstruction phases, we use standard error-correcting codes to encode each message exchanged between parties, following the communication pattern of the GMW protocol. Using an ECC of length $O_\varepsilon(\kappa + \log \eta)$, where $\eta$ denotes the total number of input and output wires in the circuit, ensures that each decoded message in these phases is correct with probability at least $1 - \mathsf{negl}(\kappa)$.
2. For the evaluation phase, we apply our novel IC scheme (Theorem 6) to simulate a variant of the GMW protocol in which the required OT correlations are first generated using the protocol of [HIKN08] and then consumed during the circuit evaluation.

The correctness of our MPC protocol reduces to the correctness of the classic GMW protocol (Section 4.2). This follows from the guarantees of our IC scheme, which ensures that all parties simulate each round of the GMW protocol synchronously within the same simulation iteration. As a result, all parties use consistent instances of correlated randomness. Moreover, since the underlying IC scheme is constant-rate, the overall communication complexity of the resulting noise-resilient MPC protocol is $O_\varepsilon(n^2(|C| + \kappa))$.

The privacy proof is more involved. In Theorem 8, we show that a variant of the GMW $\mathsf{NAND}$ protocol (encoded via the $\mathsf{A2A}$ scheme), while not always correct due to the noisy communication, satisfies a suitable notion of privacy: there exists a simulator whose output, given the corrupted

parties' inputs, is indistinguishable from the view of the protocol over noisy channels. We combine this simulator with the simulator of the classic (noiseless) GMW NAND protocol to construct a simulator for the full MPC protocol.

Intuitively, the MPC simulator invokes the noisy NAND simulator independently for each chunk, except for chunks corresponding to output gates that are correctly evaluated by all parties. In those cases, the simulator instead uses the classic GMW NAND simulator, given the corrupted inputs and the ideal MPC output. This overview omits several technical components, including how the simulator generates views consistent with the exchanged tree-code symbols. A complete description of the simulator and the full privacy proof appears in Section 4.

### 1.2 Paper Overview

We begin in Section 2 by introducing notation and basic definitions (additional preliminaries appear in Section B and additional related work appears in Section C). In Section 3, we present our new coding schemes; due to page limitations, their analysis and further technical details are deferred to Section D and Section E. Next, in Section 4, we present our noise-resilient MPC construction, with its analysis and additional details deferred to Section G. Further extensions and optimizations of our main result (e.g., to other noisy channels) are discussed in Section H. Finally, we conclude and outline several directions for future work in Section I.

## 2 Preliminaries

*Notations.* For any integer $k \in \mathbb{N}$, let $[k]$ represent the set $\{1, 2, \ldots, k\}$. The notation $\leftarrow_{\$} S$ and $\leftarrow \mathcal{D}$ signifies sampling at random from the set $S$ and the distribution $\mathcal{D}$, respectively. $\mathsf{bernoulli}(p)$ represents the standard Bernoulli distribution with parameter $p \in [0, 1]$, yielding 1 with probability $p$ and 0 with probability $1 - p$. Given any finite set (alphabet) $\Sigma$, let $\Sigma^*$ denote the set of all possible strings of finite length over the alphabet $\Sigma$. Given any $x \in \Sigma^*$, we let $\mathrm{len}(x) = |x|$ denote the length of $x$. For any vector (or string) $x$, the notation $x[i]$ denotes the $i$-th component of $x$ and $x[i, j]$ with $1 \leqslant i < j \leqslant |x|$ denotes the respective sub-vector or sub-string. Note that $x[i]$ can itself be a string or a vector. For any set $S \subseteq [n]$ and vector $x$ of length $n$, we let $x_S = \{x[i] \mid i \in S\}$. For a matrix $M$, $M_i$ is its $i^{th}$ row and $M^i$ is its $i^{th}$ column.

For functions $f, g$, the notation $f(n, m) = O_m(g(n))$ indicates that the function $f$ grows at most at the order of $g$ when $m$ is treated as a constant. To simplify notation, whenever it is clear from the context, we might write $O(\cdot)$ instead of $O_\varepsilon(\cdot)$, etc.

*Network and Protocols.* We consider a fully-connected network with $n$ parties, $P_1, \ldots, P_n$. The parties execute a synchronous protocol where in each round of the protocol each pair of parties exchange a single bit over a private channel connecting them. The parties' goal is to evaluate some fixed and known-by-all $n$-party function $f : \{\{0,1\}^m\}^n \to \{\{0,1\}^{m'}\}^n$ on their private inputs: $P_i$ starts with inputs $x_i \in \{\{0,1\}^m\}$, and obtains the $m'$-bit output $f_i(x)$, where $x = (x_1, \ldots, x_n)$ and $f(x) = (f_1(x), \ldots, f_n(x))$.

To simplify the protocol description, we assume that the network has self-loops, and messages sent by a party to itself are not considered to be sent over the network.

In this work, all communication channels are prone to stochastic noise. In particular, each channel is a binary symmetric channel (BSC) that flips each bit independently with some fixed probability $\varepsilon \in (0, 1/2)$.

*Correlated inputs.* We consider protocols where parties also take as input a vector of *correlations*; these can be specified by a set $\mathbb{C}$, where each element of the set is a vector of size $n$ that describes for each party the components of the correlations that party obtains. If a protocol uses $t$ instances of $\mathbb{C}$ correlations, then we represent its vector of correlations by $\mathsf{corr} \in \mathbb{C}^t$. We can view the vector of vectors $\mathsf{corr}$ as a matrix of size $t \times n$. Hence, $\mathsf{corr} = (\mathsf{corr}^1, \ldots, \mathsf{corr}^n)$, where $\mathsf{corr}^i$ represents the $t$ correlation of $P_i$.

Let us define a specific set of correlations used in this paper, which we call random-OT ($\mathsf{ROT}$) correlations, and denote by $\mathbb{C}_{\mathsf{ROT}}$. Each correlation is shared between two parties, where one of them is the *sender* and the other is the *receiver*. $\mathbb{C}_{\mathsf{ROT}}$ contains all elements of the form $((r_0, r_1), (b, r_b))$, where $r_0, r_1, b \in \{0, 1\}$, where the sender's and receiver's correlation components are $(r_0, r_1)$ and $(b, r_b)$, respectively. In our protocols, we will use $n$-party correlations that contain $\mathbb{C}_{\mathsf{ROT}}$ correlations for every pair of parties, where the party with the lower index acts as the sender and the party with the higher index acts as the receiver. For each $t$ rounds of our protocol, for $t$ some small constant, the parties use a new correlations vector. In Appendix B we give formal definitions that encapsulates the correctness of a protocol with respect to input correlations.

### 2.1 Interactive Coding

Recall that $n$ parties run a synchronous protocol, where in each round they exchange a single bit. It is sometimes easier to assume that the parties communicate symbols of some alphabet $\Sigma_C$; each symbol can be communicated in $\log |\Sigma_C|$ rounds by its bit-representation. We use the following notations. For any round $r \geqslant 1$, let $m_{\to i}(r) \triangleq (m_{1\to i}(r), \ldots, m_{n\to i}(r))$ be all the messages (symbols of $\Sigma_C$) received by $P_i$ until round $r$ (inclusive). We assume $m_{j\to i}(0)$ and $m_{i\to i}(r)$ to be empty for any $i, j, r$. We similarly define $m_{i\to}(r) \triangleq (m_{i\to 1}(r), \ldots, m_{i\to n}(r))$ as all the symbols sent by $P_i$ up to round $r$, inclusive. To summarize, in the execution of the protocol $\pi$, party $P_i$ uses $\pi(x_i, m_{\to i}(r))$ to compute the symbol it wishes to send to each $P_j$ in round $r + 1$, records it in $m_{i\to}$, receives a symbol from each $P_j$, records it in $m_{\to i}$, and so on. The total number of rounds of $\pi$, denoted $|\pi|$, is also referred to as its *round complexity*. We shall refer to $m_{i\to}(|\pi|)$ and $m_{\to i}(|\pi|)$ respectively as the *sent and received transcripts* of $P_i$.

*Interactive coding schemes.* When $\pi$ is designed to work over noiseless channels, but each link in the network is a noisy binary symmetric channel $\mathsf{BSC}_\varepsilon$, the parties can use an *interactive coding scheme* (IC) that allows them to simulate $\pi$ over the noisy network. The IC scheme generates from $\pi$ a new protocol, $\tilde{\pi}$, such that each $P_i$ can deterministically reconstruct the transcript $m_{i\to}(|\pi|)$ and $m_{\to i}(|\pi|)$ based on $x_i$ and the sent and received transcripts of $P_i$ for $\tilde{\pi}$, with high probability. An interactive coding scheme has *constant rate* if $\mathtt{CC}(\pi') = O(\mathtt{CC}(\pi))$, when $|\pi| \to \infty$.[5]

The following well-known result states that an interactive coding with a constant blow-up in the number of rounds exists.

**Theorem 2 ([ABE+19]).** *There exists an $n$-party simulation $\tilde{\pi}$ for every fully-utilized protocol $\pi$ on a clique, with $|\tilde{\pi}| = O(|\pi|)$ and probability of error $2^{-\Omega(|\pi|\sqrt{n})}$.*

The simulation $\tilde{\pi}$ of Theorem 2 builds upon two important ingredients: (a) the standard Rajagopalan-Schulman (RS) scheme of [RS94] acting as an outer code, and (b) the $\mathsf{A2A}$ *scheme* of [ABE+19]

[5] Note that this differs from the rate of MPC protocols (defined in Section 2.2 below), which measures the communication complexity of the MPC protocol with respect to the size of the Boolean circuit it is evaluating.

acting as an inner code. Both the RS scheme and the A2A scheme are outlined in more detail and formally described in Appendix B.1 for self containment. We briefly explain their role here, as we will use these primitives in our construction.

The A2A scheme solves the following bit-exchange task. Assume that each party $P_i$ holds $n-1$ bits $b_i = (b_{i,1}, \ldots, b_{i,n})$, where $b_{i,j}$ needs to be communicated to $P_j$. To accomplish this, the parties are allowed to run a fully-utilized protocol on the clique with noisy links and reconstruct the $n-1$ bits they need to learn based on their respective sent and received transcripts. The bit-exchange task is successful if all the parties $P_i$ are able to correctly recover the bits $b_{j,i}$ for $j \in [n]$. The A2A scheme is an $O(1)$-round protocol that accomplishes the broadcast task with a vanishing probability of error, as summarized in the theorem below.

**Theorem 3 ([ABE+19]).** *The* A2A *scheme performs the bit-exchange task, takes O(1) rounds and succeeds except with probability* $2^{-\Omega(\sqrt{n})}$.

The RS scheme allows simulating with $O(1)$ overhead, any protocol $\pi$ over arbitrary noisy networks, assuming BSC links with sufficiently small flip probability $\varepsilon$. The A2A provides this sufficiently small flipping overhead over a clique, and together, the concatenated scheme achieves simulation of any $\pi$ with $O(1)$ overhead with high probability over arbitrary $\mathsf{BSC}_\varepsilon$ channels with any $\varepsilon < 1/2$.

Our construction also uses standard error-correcting codes ECC [Sha48]. This code allows us to encode any message with an additive overhead of $O(\kappa)$ bits and ensure that decoding fails over BSC with probability $\leqslant 2^{-\kappa}$. A formal description of such codes is deferred to Appendix B.1.

### 2.2 Secure Multiparty Computation

In the context of general secure multiparty computation (MPC), We are given a Boolean function, represented as circuit $C$ which the parties wish to evaluate in a secure (private) way. We consider circuits consisting only of NAND gates; these compute the negation of the logical AND of their input bits and form a complete basis for Boolean computation. A gate in the circuit is called an output gate if its output wire corresponds to one of the outputs of the circuit $C$. The size of the circuit $|C|$ corresponds to the number of NAND gates in the circuit.

*MPC security over noiseless channels.* We consider standard simulation-based statistical semi-honest security for $n$ parties based on the real-world/ideal-world paradigm of [Can00]. Parties participate in an execution of $\pi$ over noiseless channels in the presence of an adversary Adv who corrupts a set $\mathcal{A}$ of parties. Define the view of a set of parties $\mathcal{A}$, $\mathsf{View}_\pi(\mathcal{A}; x, \kappa) = \{x_\mathcal{A}, r_\mathcal{A}, \mathsf{corr}^\mathcal{A}, T_\mathcal{A}\}$, as their inputs $x_\mathcal{A}$, any internal randomness $r_\mathcal{A}$ used by them, their correlated randomness $\mathsf{corr}^\mathcal{A}$, and their *transcript*, $T_\mathcal{A}$, namely, the set of their received messages in the protocol. $\kappa$ is a statistical security parameter. Also let $\mathsf{Output}_\pi(x, \kappa)$ represent the output of all parties in the execution. We also define the augmented view of the corrupt parties ($\mathsf{View}^*_\pi$) to include the transcript of the honest parties, alongside the output of $\mathsf{View}_\pi(\mathcal{A}; x, \kappa)$.

A real-world protocol $\pi$ is said to be secure against semi-honest adversaries if whatever can be learned from the real-world adversary can be learned from its inputs and outputs alone. That is, a protocol $\pi$ is secure if there exists an efficient simulator Sim which, given the security parameter, inputs and outputs of the corrupt parties, can generate the view of the real-world adversary Adv.

**Definition 1 (Semi-Honest Security over Noiseless Channels).** *An $n$-party protocol $\pi$ for computing a function $f$ is said to be statistically secure against semi-honest adversaries over noiseless channels if, for any $\mathcal{A} \subset [n]$, for every real-world adversary corrupting a set $\mathcal{A}$ of parties, there exists an efficient simulator* $\mathsf{Sim}$ *such that, for all inputs $x$ and statistical security parameter $\kappa$, the following holds:*

$$\left\{\mathsf{Sim}(\mathcal{A}; 1^\kappa, x_\mathcal{A}, f_\mathcal{A}(x)), f(x)\right\} \approx_{\exp(-\kappa)} \left\{\mathsf{View}_\pi(\mathcal{A}; 1^\kappa, x), \mathsf{Output}_\pi(1^\kappa, x)\right\}$$

Further, we also consider a stronger security notion called *semi-honest security with channel transparency*, where the simulator can generate the augmented view of the corrupted parties, such that the joint view of the augmented view and the protocol output are indistinguishable from the simulator output and the ideal output. We say that an MPC protocol $\pi$ for computing a function represented by a Boolean circuit $C$ has *linear communication overhead* if its communication complexity scales linearly in the size of the circuit $|C|$.

*Oblivious Transfer.* The Oblivious Transfer ($\mathsf{OT}$) [Rab81, EGL82] functionality $\mathcal{F}_\mathsf{OT}$ is a 2-party functionality between two parties, a sender and a receiver. The input of the sender consists of two bits $(x_0, x_1)$, and the receiver's input is a single choice bit $b$. The output of the functionality $\mathcal{F}_\mathsf{OT}((x_0, x_1), b)$ is $(\perp, x_b)$, where the receiver learns the bit $x_b$ while learning nothing about $x_{1-b}$, and the sender learns nothing about the value $b$. A related functionality is Random Oblivious Transfer ($\mathcal{F}_\mathsf{ROT}$), where the parties provide no inputs, and the functionality outputs a random "instance" of $\mathsf{OT}$, namely the outputs $(x_0, x_1)$ and $(b, x_b)$, respectively, where $x_0, x_1, b \leftarrow_\$ \{0,1\}$.

*MPC security over noisy channels.* The MPC definition for noiseless networks extends naturally to the case where all communication channels are noisy, with the caveat that parties may now have inconsistent views of the transmitted and received messages. For any set of corrupt parties $\mathcal{A}$ and inputs $x$, define the view over $\mathsf{BSC}_\varepsilon$ as $\mathsf{View}_\pi^\mathsf{BSC}(\mathcal{A}; x, \kappa) = \{x_\mathcal{A}, r_\mathcal{A}, \mathsf{corr}^\mathcal{A}, T_\mathcal{A}^\mathsf{BSC}\}$, where $T_\mathcal{A}^\mathsf{BSC}$ denotes the messages received by corrupt parties as delivered through the noisy channels, and the remaining symbols carry the same meaning as in the definition of $\mathsf{View}_\pi$. The corresponding output of parties over $\mathsf{BSC}$ is represented by $\mathsf{Output}_\pi^\mathsf{BSC}$. The formal definition of statistical security in this setting, together with the stronger security notion of *semi-honest security with channel transparency over* $\mathsf{BSC}$ can be found in Appendix G.1.

*Random OT over* $\mathsf{BSC}$ *channels.* The generic MPC protocol we construct (Theorem 11) is in the $\mathcal{F}_\mathsf{ROT}$-hybrid model over $\mathsf{BSC}_\varepsilon$ channels. We instantiate the $\mathcal{F}_\mathsf{ROT}$ functionality by utilizing the construction from [HIKN08], which produces $O(\ell)$ many $\mathsf{ROT}$ correlations between any two parties, communicates over $\mathsf{BSC}_\varepsilon$ channels with communication complexity $O_\varepsilon(\ell)$ bits, and features semi-honest security.

**Theorem 4 ([HIKN08, Theorem 9]).** *For any constant $\varepsilon \in (0, 1/2)$ and any $\ell$, there exists a two-party protocol $\pi^{OT^\ell}$ that securely realizes $\mathcal{F}_\mathsf{ROT}$ functionality over $\mathsf{BSC}_\varepsilon$ channels. The protocol is statistically secure against semi-honest parties with error $2^{-\Omega(\ell)}$ and communication complexity of $O_\varepsilon(\ell)$ bits.*

## 3 A New Multi-Party Interactive Coding Scheme

In order to design our noise-resilient MPC protocol, we need a multiparty IC scheme that (i) works on continuous chunks of $c$ rounds instead of a round-by-round fashion; and (ii) simulates each round

of the underlying noiseless protocol using the same correlated randomness. To ease the reading, we split the construction into two independent parts: a constant-rate IC scheme that operates in chunks is presented and analyzed in Section D, and a constant-rate IC scheme with simultaneous simulation, which supports protocols that use correlated randomness, is presented in Section 3.1 (with its analysis in Section E). Finally, in Section 3.5 (and Appendix F), we combine these two constructions into a single constant-rate IC scheme that satisfies both properties simultaneously.

### 3.1 Interactive Coding Scheme with Simultaneous Simulation

In this section, we construct a variant of the schemes in [ABE+19, RS94] for noisy cliques with an additional property we call *simultaneous simulation*: whenever a party $P_i$ simulates a round of $\pi$, all other parties simulate the same round of $\pi$. This property is crucial when $\pi$ relies on correlated inputs, such as round-based shared randomness, as it ensures that all parties use the same correlated randomness when simulating the same round. Our goal is the following.

**Theorem 5.** *Given a clique with $n$ parties connected via pairwise $\mathsf{BSC}_\varepsilon$ with $\varepsilon < 1/2$, for any (noiseless) protocol $\pi$ there exists a protocol $\tilde{\pi}$ that perform a simultaneous simulation of $\pi$ over the noisy network in $O_\varepsilon(|\pi|)$ rounds and succeeds except with probability $2^{-\Omega_\varepsilon(|\pi|\sqrt{n})}$ over the random noise.*

We restrict attention to sufficiently long protocols, i.e., $|\pi| = \omega(n)$, since otherwise one could simply simulate each round using the A2A scheme and obtain a comparable guarantee.[6]

### 3.2 Overview

The simulation $\tilde{\pi}$ communicates symbols from a quaternary alphabet $S \triangleq \{0, 1, \mathrm{BK}, \mathrm{H}\}$. To describe the simulation, we shall make use of a tree code $\mathcal{T} : S^* \to \Sigma^*$ of distance $\alpha = \frac{1}{2}$, see Appendix B.1 for a formal definition. To simplify notations, we will assume that the channel alphabet is also $\Sigma$, i.e., in any round of the simulation parties exchange a single tree-code symbol on each link. Received symbols may be corrupted in a manner that is equivalent to looking on the $\log|\Sigma|$ bits describing a single symbol, and flipping each bit independently with probability $\varepsilon$.

Fix $\pi$ and let $R = |\pi|$. We now give an overview of the simulation, formally depicted in Algorithm 1. The simulation consists of $3R$ iterations,[7] divided into two phases. The first phase, referred to as the *information phase*, consists of $2R$ iterations of actual information exchange between the parties. This phase is followed by a *dummy phase* of $R$ iterations. The purpose of the dummy phase is to ensure that the parties correctly decode the tree-code symbols of the information phase.

For each $j \in [n]$, party $P_i$ maintains two strings $s_{i,j}, w_{i,j} \in S^*$, recording the symbols it has sent to and received from $P_j$, respectively. More precisely, $s_{i,j}$ is the transcript of symbols sent by $P_i$, and $w_{i,j}[1, r-1]$ records the symbols that $P_i$ believes $P_j$ has sent; these were used to generate $P_i$'s outgoing symbols $s_{i,j}[1, r]$. In addition, $P_i$ maintains the string $\hat{w}_{i,j}$, recording the symbols actually received from $P_j$. Note that these may differ from $w_{i,j}$ due to errors. In a sense, $w_{i,j}$ is $P_i$'s estimate of the symbols sent by $P_j$, based on the possibly corrupted received $\hat{w}_{i,j}$.

[6] We encourage the reader to check Appendix B.1 for a brief description of previous IC schemes and related techniques on which our construction is based.

[7] In order to distinguish rounds of $\pi$ from rounds of the simulation $\tilde{\pi}$ we will call the former *rounds* and the later *iterations*.

**Algorithm 1** Multi-party Coding Scheme with simultaneous simulation (actions taken by $P_i$)

**inputs:** a. A protocol $\pi$ with zero-padding after $R \triangleq |\pi|$ transmissions; b. An input $x_i$; c. A tree-code $\mathcal{T} : S^* \to \Sigma^*$ with distance $\alpha = \frac{1}{2}$.

**initialize** $s_{i,j}, w_{i,j} \leftarrow \emptyset, \forall j \neq i$

**for** $r = 1$ to $2R$ **do**
  $\hat{w}_{i,j} \leftarrow$ the decoding of the tree-coded string (of length $r-1$) received from $P_j$ so far
  $(\sigma_{i,j}, \hat{z}_{i,j}, z_{i,j}) \leftarrow \text{PARSE}(s_{i,j}, \hat{w}_{i,j}, w_{i,j})$
  flag $\leftarrow \text{ConsCheck}(\{\sigma_{i,j}, \hat{z}_{i,j}, z_{i,j} \mid j \neq i\})$

  $l \leftarrow |\sigma_{i,j}|$ for some $j \neq i$ ▷ Any choice of $j$ suffices
  $\mathsf{Ind}_i \leftarrow \text{GenIndex}(s_{i,j})$ for some $j \neq i$ ▷ Any choice of $j$ suffices
  $\mathsf{Ind}_{i,j} \leftarrow \text{GenIndex}(\hat{w}_{i,j})$
  ▷ Simulate the next round
  **if** flag $= 1$ **and** $\mathsf{Ind}_i[r-1] = l$ **and** for all $j \neq i$, $\mathsf{Ind}_{i,j}[r-1] = l$ **then**
    **for** all $j \neq i$ (in parallel) **do**
      $w_{i,j}[r-1] \leftarrow \mathsf{Info}(\hat{z}_{i,j}[l])$ ▷ "Record" used received symbol
      $temP_j \leftarrow \mathsf{Info}(z_{i,j}) \circ w_{i,j}[r-1]$
      $(s_{i,1}[r], \ldots, s_{i,n}[r]) \leftarrow \pi(x_i, \{temP_j\}_{j \in [n]})$ ▷ next round of $\pi$
      Send $\mathcal{T}(s_{i,j})[r]$ to $P_j$ using $\mathsf{A2A}$
    **end for**
  ▷ Some neighbour is ahead, hold
  **else if** flag $= 1$ **and** $\mathsf{Ind}_i[r-1] = l$ **and** for all $j \neq i$, $\mathsf{Ind}_{i,j}[r-1] \geqslant l$ **then**
    **for** all $j \neq i$ (in parallel) **do**
      $w_{i,j}[r-1] \leftarrow \text{H}$
      $s_{i,j}[r] \leftarrow \text{H}$
      Send $\mathcal{T}(s_{i,j})[r]$ to $P_j$ using $\mathsf{A2A}$
    **end for**

  **else** ▷ Inconsistency found, rewind
    **for** all $j \neq i$ (in parallel) **do**
      $w_{i,j}[r-1] \leftarrow \text{BK}$
      $s_{i,j}[r] \leftarrow \text{BK}$
      Send $\mathcal{T}(s_{i,j})[r]$ to $P_j$ using $\mathsf{A2A}$
    **end for**
  **end if**
**end for**

**for** $r = 1$ to $R$ **do** ▷ Dummy Phase, send more redundancy
  Send $\mathcal{T}(s_{i,j} \circ 0^r)[2R + r]$ to $P_j$ using $\mathsf{A2A}$
**end for**

$\hat{w}_{i,j} \leftarrow$ the decoding of the tree-coded string (of length $3R$) received from $P_j$ so far
For all $j \neq i$: $(\sigma_{i,j}, \hat{z}_{i,j}, z_{i,j}) \leftarrow \text{PARSE}(s_{i,j}, \hat{w}_{i,j}, w_{i,j})$
**return** $(\sigma_{i,j}, z_{i,j})_{j \neq i}$ with all strings truncated to the first $R$ indices

Very roughly, the actions taken by $P_i$ in iteration $r$ of the simulation are as follows. For every $j \neq i$, party $P_i$ decodes all the tree-code symbols received from $P_j$ so far (i.e., $r-1$ such symbols) and obtains the string $\hat{w}_{i,j} \in S^{r-1}$. Then, it checks whether (1) the simulation so far is consistent with $\pi$ and (2) the simulation is simultaneous. This is performed via the ConsCheck() procedure stated in Algorithm 4, based on $P_i$'s knowledge about symbols sent to and received from other parties, i.e., $s_{i,j}, w_{i,j}$ for $j \neq i$, and the freshly decoded $\hat{w}_{i,j}$. Note that the "recorded symbols" string $w_{i,j}$ always lags one symbol behind $s_{i,j}, \hat{w}_{i,j}$, (i.e., it is of length $r-2$); for all $P_i$ computations that involve the received transcript from $P_j$, the first $r-2$ symbols are taken from $w_{i,j}$, while the $(r-1)$-th symbol is taken from the freshly decoded $\hat{w}_{i,j}$.

If ConsCheck indicates that the simulation is not consistent with $\pi$ or that it is not simultaneous, $P_i$ assigns $s_{i,j}[r] = \text{BK}$ for all $P_j$, which indicates it wishes to remove the last simulated round from the parties' recorded transcripts. Additionally, $P_i$ sets $w_{i,j}[r-1] = \text{BK}$ for all $P_j$s to delete the record of the symbols received from $P_j$s in the previous iteration. This is because a failed consistency check supposedly indicates that either the previous round was not simultaneously simulated, or the recorded received symbols from some $P_j$ in the previous iteration was incorrect. If the simulation is consistent and simultaneous, but $P_i$ believes that at least one of its neighbours have simulated at least one extra round, then $P_i$ decides to hold its current position (i.e., it does not continue to simulating the next round of $\pi$). Technically, it assigns $s_{i,j}[r] = w_{i,j}[r-1] = \text{H}$ for all $P_j$s. Finally, if $P_i$ believes that the simulation so far has been simultaneous and consistent, and every neighbour has simulated exactly the same number of rounds, then it decides to simulate the next round of $\pi$. To that end, it sets $s_{i,j}[r]$ according to the next symbol to be sent according to $\pi$. It also assigns $w_{i,j}(r-1) = \hat{w}_{i,j}(r-1)$, recording the last received symbol for the next iteration.

After updating the strings $w_{i,j}$ and $s_{i,j}$ according to the above three scenarios, the $P_i$ computes the next tree-code symbol $\mathcal{T}(s_{i,j})[r]$ and communicates it to $P_j$ (algorithm 1). The exchange of all these symbols is performed simultaneously via the A2A scheme (Figure 5).

### 3.3 Consistency Check, Parse, and Index Generation

It remains to describe how a party checks whether consistent and simultaneous simulation has occurred so far. To check simultaneous simulation, $P_i$ needs to keep track of the round of the original protocol $\pi$ that it simulated in any given iteration of the simulation protocol. This is done through the GenIndex procedure detailed in Algorithm 2. The GenIndex procedure takes any of the strings $s_{i,j}, \hat{w}_{i,j}$ as input, and returns, for each element in $s_{i,j}$ or $\hat{w}_{i,j}$, the round of $\pi$ that corresponds to that element in the simulation. The procedure works as follows. The output will be given via a string of round numbers Index, initialized to the all zeros string, and updated serially starting from the first location. At index location $l$, if $s_{i,j}[l] \in \{0,1\}$, then the algorithm assigns the value $\text{Index}[l-1]+1$ to $\text{Index}[l]$ (assuming $\text{Index}[0] = 0$). If $s_{i,j}[l] = \text{H}$, then $\text{Index}[l]$ is assigned the value $\text{Index}[l-1]$. This indicates that in the $l$-th iteration, the party $P_i$ has decided to halt the simulation and wait for other parties to rewind back to the current simulated round. Finally, if $s_{i,j}[l] = \text{BK}$, then $\text{Index}[l]$ is assigned with the value $\text{Index}[l]-1$, indicating that the last simulated round has been deleted. Note that the output of $\text{GenIndex}(s_{i,j})$ is independent of the choice of the neighbour $P_j$ as Algorithm 1 simultaneously assigns H or BK to all strings $s_{i,j}$ in a given simulation round. On the other hand $\text{GenIndex}(\hat{w}_{i,j})$ may differ between different $P_j$s.

Next, $P_i$ deduces how many rounds of $\pi$ have been simultaneously simulated so far, and the simulation iteration they have been generated in. To do so, it executes the PARSE procedure, which is detailed in Algorithm 3. The PARSE procedure takes as inputs the strings $s_{i,j}, w_{i,j}, \hat{w}_{i,j}$

**Algorithm 2** The GenIndex procedure

**input:** $s \in S^l$
**initialize:** Index $\leftarrow 0^l$, $i \leftarrow 0$

**for** $j = 1$ to $l$ **do**
  **if** $s[j] \in \{0, 1\}$ **then**
    Index$[j] \leftarrow i + 1$
    $i \leftarrow i + 1$
  **else if** $s[j] =$ H **then**
    Index$[j] \leftarrow i$
  **else** ($s[j] =$ BK)
    Index$[j] \leftarrow \max\{i - 1, 0\}$ ▷ Do not rewind below 0
    $i \leftarrow$ Index$[j]$
  **end if**
**end for**

**return** Index$[1, l]$

for every $P_j$, and generates respectively three sets of parsed strings $\sigma_{i,j}$, $z_{i,j}$, $\hat{z}_{i,j}$. The parsed strings include only undeleted simulated rounds (i.e., they do not include the control directives H and BK, nor rounds that were rewound due to a BK command), and for each such simulated rounds, they also include the iteration in which that round was simulated, according to the GenIndex output.

The PARSE procedure behaves as follows. A counter for the last simultaneously simulated round, denoted by $\mathsf{minInd}$ is initialized to 0. Now, for every round $l \leqslant r - 1$, the algorithm checks whether indices $\mathsf{Ind}_i[l]$ and $\mathsf{Ind}_{i,j}[l]$ all equal $\mathsf{minInd} + 1$, and whether all the strings $s_{i,j}$ and $\hat{w}_{i,j}$ hold information symbols at location $l$. If so, this means that a new round was simultaneously simulated at round $l$. Accordingly, $z_{i,j}, \sigma_{i,j}, \hat{z}_{i,j}$ are appended respectively with $(w_{i,j}[l], l), (s_{i,j}[l], l)$, and $(\hat{w}_{i,j}[l], l)$; the first item in each tuple is the simulated information bit and the second item is the iteration in which this bit was simulated. Also, $\mathsf{minInd}$ is increased by 1. Otherwise, if it turns out that $\mathsf{Ind}_i[l] < \mathsf{minInd}$, this implies that some rounds which were previously thought to have been simultaneously simulated were deleted. Accordingly, the last $\mathsf{minInd} - \mathsf{Ind}_i[l]$ symbols from each of the parsed strings $z_{i,j}, \sigma_{i,j}, \hat{z}_{i,j}$, for all $j \neq i$ are deleted. Also, $\mathsf{minInd}$ is set to $\mathsf{Ind}_i[l]$, which the party $i$ believes to be last simultaneously simulated rounds. After its completion, the algorithm returns the parsed strings $z_{i,j}, \sigma_{i,j}, \hat{z}_{i,j}$, for all $j \neq i$.

To complete the description of the consistency check, we first introduce some notation. Recall that the $l$-th index of a parsed string (say, $\sigma_{i,j}$) contains a tuple $(a, b)$, where the first element is an information bit and the second element is an iteration number. We denote these two, respectively, by $\mathsf{Info}(\sigma_{i,j}[l])$ and $\mathsf{Rnd}(\sigma_{i,j}[l])$.

After obtaining the parsed strings (algorithm 1 in Algorithm 1), $P_i$ checks whether they are all consistent with protocol $\pi$, and whether simultaneous simulation has occurred so far. This is performed by the ConsCheck() procedure depicted in Algorithm 4. This procedure simply checks whether the parsed string of all symbols received so far, i.e., $\hat{z}_{i,j}$, matches with the parsed recorded received transcript $z_{i,j}$ (used to generate previous sent symbols), for all $P_j$s. We note that unlike the original scheme in [RS94], we do not have to check whether $|\hat{z}_{i,j}| > |z_{i,j}|$, since the strings are parsed simultaneously by the PARSE procedure, which ensures that $|\hat{z}_{i,j}| = |z_{i,j}| + 1$, with the one extra symbol being the latest received information symbol. Furthermore, we have $\mathsf{Info}(\sigma_{i,j}[l]) = \pi(x_i, \{\mathsf{Info}(z_{i,j}[1, l-1]\}_{j \neq i})$ by default, directly from Algorithm 1.

**Algorithm 3** The PARSE procedure (for $P_i$)

**input:** For all $j \neq i$: $s_{i,j} \in S^{r-1}$, $\hat{w}_{i,j} \in S^{r-1}$, $w_{i,j} \in S^{r-2}$ (for some $r \geqslant 1$)

▷ $S^k := \{\emptyset\}$ for $k \leqslant 0$

**initialize:** $\hat{z}_{i,j}, z_{i,j}, \sigma_{i,j} \leftarrow \emptyset$, $\mathsf{minInd} \leftarrow 0$, $l \leftarrow 1$

$\mathsf{Ind}_i \leftarrow \text{GenIndex}(s_{i,j})$ for some $j \neq i$ ▷ Any choice of $j$ suffices

$\mathsf{Ind}_{i,j} \leftarrow \text{GenIndex}(\hat{w}_{i,j})$, $\forall j \neq i$

**while** $l \leqslant r-1$ **do**
  **if** for all $j \neq i$: $\mathsf{Ind}_i[l] = \mathsf{Ind}_{i,j}[l]$ **then**
    **if** $\mathsf{Ind}_i(l) = \mathsf{minInd} + 1$ **and** for all $j \neq i$: $\hat{w}_{i,j}[l], s_{i,j}[l] \in \{0,1\}$ **then**
      $\hat{z}_{i,j} \leftarrow \hat{z}_{i,j} \circ (\hat{w}_{i,j}[l], l)$
      $\sigma_{i,j} \leftarrow \sigma_{i,j} \circ (s_{i,j}[l], l)$
      **if** $l \geqslant 2$ **then** $z_{i,j} \leftarrow z_{i,j} \circ (w_{i,j}[l-1], l-1)$ **end if**
      $\mathsf{minInd} \leftarrow \mathsf{minInd} + 1$
    **else if** $\mathsf{Ind}_i[l] < \mathsf{minInd}$ **then**
      Truncate the last $(\mathsf{minInd} - \mathsf{Ind}_i[l])$ symbols of $z_{i,j}, \hat{z}_{i,j}, \sigma_{i,j}$, for all $j \neq i$.
      $\mathsf{minInd} \leftarrow \mathsf{Ind}_i[l]$
    **end if**
  **end if**
  $l \leftarrow l + 1$
**end while**

**return** $\sigma_{i,j}, \hat{z}_{i,j}, z_{i,j}$, for all $j \neq i$

If the consistency check fails, $P_i$ sends BK to all its neighbors in this iteration (algorithms 1 to 1). If, however, the test passes, then $P_i$ checks for simultaneous simulation. To do so, it again executes GenIndex and obtains $\mathsf{Ind}_i$ and $\mathsf{Ind}_{i,j}$ (algorithm 1; these were already computed in PARSE). Now, $P_i$ checks if all parties are set to simulate the same round of $\pi$ , namely, whether $\mathsf{Ind}_i[r-1] = \mathsf{Ind}_{i,j}[r-1] = |\sigma_{i,j}|$ for all $j \neq i$. In this case, $P_i$ simulates the next round of $\pi$ (algorithm 1), and otherwise, it either holds (algorithm 1) or rewinds (algorithm 1). When $\mathsf{Ind}_i[r-1] = |\sigma_{i,j}|$ with all neighbors $P_j$ not being behind, $\mathsf{Ind}_{i,j}[r-1] \geqslant |\sigma_{i,j}|$, and at least one $P_j$ being ahead satisfying $\mathsf{Ind}_{i,j}[r-1] > |\sigma_{i,j}|$, party $P_i$ holds at its current position and sends H to all its neighbours (who will in turn rewind to the longest shared simultaneous prefix). If none of these hold (i.e., at least one $P_j$ is behind) $P_i$ rewinds towards that round by sending BK to all its neighbours, indicating that it is deleting the last simulated step.

**Algorithm 4** The ConsCheck procedure (for $P_i$)

**inputs:** For all $j \neq i$: $\sigma_{i,j}, \hat{z}_{i,j}, z_{i,j}$

**initialize:** flag $\leftarrow 1$

$l \leftarrow |\sigma_{i,j}|$ for some $j \neq i$ ▷ Any choice of $j$ suffices

**for** $q = 1$ to $l-1$ **do**
  **if** $\exists j \neq i$ such that $z_{i,j}[q] \neq \hat{z}_{i,j}[q]$ **then**
    flag $\leftarrow 0$ ▷ inconsistency found
  **end if**
**end for**

**return** flag

### 3.4 Dummy Phase and Output

After running $2R$ simulation iterations (algorithm 1 of Algorithm 1), the parties continue with additional $R$ *dummy* iterations (algorithm 1). In these iterations, the parties send only tree-code symbols that extend their sent transcripts with zeros. The reason for this dummy phase is to avoid errors in decoding the suffix of the tree code. Indeed, we prove in Section E.2 that, with high probability, after $2R$ iterations of the simulation, the transcript encoded by the tree code is correct. However, channel noise that occur around iteration $2R$ could prevent parties from decoding the complete string, so that the suffix of the string is corrupted (see suffix distance properties of tree codes in [Gel17]). The dummy phase guarantees that even if a suffix of length $R-1$ is corrupted (which can happen with negligible probability), the $2R$-long prefix can still be correctly recovered. While one could potentially just run the loop of algorithm 1 for $R$ additional iterations, adding this dummy phase greatly simplifies the analysis (See section E.2).

After the dummy phase is over, $P_i$ decodes the tree coded strings sent by all its neighbors (algorithm 1) and obtains $\hat{w}_{i,j} \in S^{3R}$. Next it parses the $2R$ first symbols (all other symbols are zeros and can be safely ignored) of $s_{i,j}, w_{i,j}, \hat{w}_{i,j}$, for all $j \neq i$ (algorithm 1). and obtains the parsed sent and received transcripts, respectively, $\sigma_{i,j}, z_{i,j}$. Finally, $P_i$ truncates each of these transcripts to length $R = |\pi|$ and outputs them all (algorithm 1).

A formal analysis of this IC scheme is presented in Appendix E.

### 3.5 Combining the Chunked Simulation with Simultaneous Simulation

It is straightforward to verify that by combining the ideas of the chunked scheme of Section D with the simultaneous simulation scheme of Section 3.1, one can obtain a simultaneous simulation scheme that simulates chunks of size $c$ bits at a time rather than a single bit. To avoid excessive repetition, we provide an overview of the merged IC scheme and the related proof technique in Appendix F, and only state here the resulting theorem.

**Theorem 6.** *For $n$ parties connected by pairwise $\mathsf{BSC}_\varepsilon$ links with $\varepsilon < 1/2$, for any constant parameter $c \in \mathbb{N}$ and for any noiseless protocol $\pi$, there exists a protocol $\tilde{\pi}$ that simulates $\pi$ in $O_\varepsilon(|\pi|)$ rounds. The simulation succeeds with probability $1 - 2^{-\Omega(|\pi|\sqrt{n})}$ over the randomness of the channel. Furthermore, the scheme $\tilde{\pi}$ works in chunks of size $c$ bits where each chunk is simultaneously simulated by all parties.*

## 4 Multi-Party Computation over BSC Channels

In this section, we first recall the original GMW construction over noiseless channels and then present our noise-resilient variant and analyze its security and communication complexity.

*The original GMW construction.* The GMW protocol [GMW87, Gol04] allows $n$ parties connected via reliable point-to-point channels to securely evaluate a circuit $C$ over their inputs. Throughout this paper, we assume that all circuits are composed solely of NAND gates. The GMW protocol consists of three phases:

– **Input Sharing Phase:** Each party $P_i$ for each input bit $x$ locally generates an $(n,n)$-additive sharing by sampling random bits $x_1, \ldots, x_n$ such that $x = \sum_{j=1}^{n} x_j$. Party $P_i$ then sends the share $x_j$ to $P_j$ for every $j \in [n]$, so that the input bit $x$ is additively and privately shared among all $n$ parties.

– **Circuit Evaluation Phase:** The parties process the circuit gate by gate in topological order. For each NAND gate with input wires carrying values $x$ and $y$ (each additively shared among the parties), they jointly compute additive secret shares of the output wire $\mathrm{NAND}(x, y) = 1 + x \cdot y$. Concretely, if $P_i$ holds shares $(x_i, y_i)$ of the input wires, the parties must compute shares $\{z_i\}$ satisfying $\sum_i z_i = 1 + (\sum_i x_i)(\sum_i y_i)$. Each party can compute the term $x_i y_i$ directly. The $x_i y_j + x_j y_i$ for $i \neq j$ are computed using two OT invocations between $P_i$ and $P_j$.
– **Output Reconstruction Phase:** After all NAND gates have been evaluated, the parties hold additive secret shares of each output wire. For each output wire, the parties exchange their shares with one another, and each party locally reconstructs the wire's value by summing the received shares.

This GMW protocol is secure in the $\mathcal{F}_{\mathsf{ROT}}$-hybrid model (recall Section 2.2) against semi-honest adversaries corrupting at most $\leqslant n - 1$ parties. Additionally, the circuit evaluation phase can be viewed as a protocol with correlations (Definition 3). In this view, the correlation index is 6, and the set of correlations consists of two instances of pairwise $\mathsf{ROT}$ correlations required for evaluating a single NAND gate.

*Noise-Resilient GMW-based protocol.* To construct our noise-resilient MPC protocol, we use a standard $\mathsf{ECC}$ code (Lemma 1) with our new IC scheme (Theorem 6). Let $C : \{\{0,1\}^m\}^n \to \{0,1\}^{m'}$ be the circuit to be evaluated, where each party provides exactly $m$ input bits. Our construction proceeds as follows:

– **Input Sharing Phase:** Each party creates $(n, n)$-additive secret shares of each of its input bits, as in the original GMW protocol. Each share is transmitted over the $\mathsf{BSC}_\varepsilon$ channel using an $\mathsf{ECC}$ with codeword length $O_\varepsilon(\kappa) + \log(mn^2)$. Since there are $\leqslant mn^2$ share transmissions (each of the $n$ parties sends $m$ shares to each of the other $n-1$ parties), and each $\mathsf{ECC}$ decoding fails with probability $2^{-(O_\varepsilon(\kappa)+\log(mn^2))}$, a union bound implies that the input sharing phase fails with probability at most $2^{-O_\varepsilon(\kappa)}$.
– **Circuit Evaluation Phase:** The parties apply the IC scheme of Theorem 6 to execute the GMW circuit evaluation phase. Crucially, the IC scheme operates with a chunk size $c = 6$, ensuring that each NAND gate evaluation (which requires 6 rounds of communication) is treated atomically. This prevents the privacy violations that would arise from rewinding mid-gate, as discussed in Section 1.1. The IC scheme guarantees that all parties simulate each gate in the same iteration, thereby ensuring consistent use of correlated randomness. Since the coding scheme has a constant rate, the communication complexity of this phase is $O_\varepsilon(n^2(|C| + \kappa))$.
– **Output Reconstruction Phase:** Each party exchanges its shares of the output wires, encoded using a standard $\mathsf{ECC}$.

A formal description of this protocol ($\Pi^{\mathsf{BSC}}_{\mathsf{MPC}}$) is provided in Figure 6 in Appendix G. The correctness of our protocol follows directly from the correctness of GMW and our IC scheme. The privacy proof, however, requires more care: we must show that even though the IC scheme may cause parties to rewind and re-execute portions of the protocol, the views of corrupt parties can still be simulated from their inputs alone. As a first step towards this proof, we analyze the security guarantee provided by a single execution of the GMW NAND protocol in the $\mathsf{ROT}$-hybrid model over the $\mathsf{BSC}_\varepsilon$ channel. Although the correctness of this protocol might fail due to channel noise, we show in the following subsection that it nonetheless satisfies a strong privacy property: the views of corrupted parties in the GMW NAND protocol over noisy channels can be simulated given only their inputs, without knowledge of the honest party inputs.

### 4.1 NAND over BSC

In this section, we analyze the privacy of the protocol $\Pi' = \mathsf{A2A}(\Pi_{\mathsf{NAND}})$, where $\Pi_{\mathsf{NAND}}$ is the GMW NAND protocol and each round of communication is encoded via the $\mathsf{A2A}$ scheme. We first describe $\Pi_{\mathsf{NAND}}$ over noiseless channels, then analyze the privacy of $\Pi'$ over $\mathsf{BSC}$ channels.

$\mathsf{NAND}$ *execution in GMW.* For the $\mathsf{NAND}$ functionality each $P_i$ inputs pair of bits $(x_i, y_i)$ for $i \in [n]$ where $\{x_i\}_i$ is a sharing of some bit $x$ and $\{y_i\}_i$ is a sharing of some bit $y$. Then the NAND of $x$ and $y$ is defined as:

$$\mathrm{NAND}(x, y) = 1 + (x_1 + x_2 + \cdots + x_n)(y_1 + y_2 + \cdots + y_n). \tag{1}$$

**Protocol $\Pi_{\mathsf{OT}}$**

**Inputs:** Party $P_i$ (the sender) holds a pair of bits $(x_0, x_1)$, and party $P_j$ (the receiver) holds a bit $b$.

1. The parties make a call to $\mathcal{F}_{\mathsf{ROT}}$: Party $P_i$ gets $x'_0, x'_1$ and Party $P_j$ gets $b', x'_{b'}$ as output.
2. Party $P_j$ sends $c = b + b'$. Assume $P_i$ receives $c'$.
3. Party $P_i$ sends $x_0 + x'_{c'}$. Assume $P_j$ receives $y_0$.
4. Party $P_i$ sends $x_1 + x'_{1-c'}$. Assume $P_j$ receives $y_1$.
5. $P_j$ outputs $y_b + x'_{b'}$.

Fig. 1: The $\Pi_{\mathsf{OT}}$ Protocol

The GMW $\mathsf{NAND}$ protocol (Figure 2) includes two OT invocations between every pair of parties. These OT invocations are implemented using the $\Pi_{\mathsf{OT}}$ protocol (defined in the $\mathcal{F}_{\mathsf{ROT}}$-hybrid model), which is depicted in Figure 1. $\Pi_{\mathsf{OT}}$ is a 3-round protocol, between a receiver $P_j$ with input $b$ and a sender $P_i$ with input $(x_0, x_1)$. The protocol $\Pi_{\mathsf{OT}}$ uses a random OT correlation, where in the first round, the receiver adjusts or derandomizes the random selection bit to match its input selection bit (by sending $c = b + b'$), and in rounds 2 and 3, the sender adjusts the OT messages by sending corrections as well. When this protocol is executed over noiseless channels, it outputs $x_b$ to the receiver and provides a simulation-based privacy guarantee in secure computation.

We now describe the protocol $\Pi_{\mathsf{NAND}}$ (presented in Figure 2) over noiseless channels in the $\mathcal{F}_{\mathsf{ROT}}$-hybrid model, which is instantiated using protocol $\Pi_{\mathsf{OT}}$ in our construction. The NAND gate computation from shares (Eq. 1) can be rewritten as $\mathrm{NAND}(x, y) = 1 + \sum_i x_i y_i + \sum_i \sum_{j \neq i} x_i y_j$.

In this expression, $P_i$ can compute the term $x_i y_i$ locally. To compute shares of the term $x_i y_j + x_j y_i$ for $i < j$, parties $P_i$ and $P_j$ do the following:

- Party $P_i$ samples random bits $r_1^{ij}, r_2^{ij}$.
- Party $P_i$, playing the role of sender, sends $(r_1^{ij}, y_i + r_1^{ij})$ to the $\mathcal{F}_{\mathsf{ROT}}$ functionality. Party $P_j$, playing the receiver, sends $x_j$ and receives $o_1^{ij}$.
- Party $P_i$, playing the role of sender, sends $(r_2^{ij}, x_i + r_2^{ij})$ to the $\mathcal{F}_{\mathsf{ROT}}$ functionality. Party $P_j$, playing the receiver, sends $y_j$ and receives $o_2^{ij}$.

By construction $(r_1^{ij} + r_2^{ij}) + (o_1^{ij} + o_2^{ij}) = x_i y_j + x_j y_i$, and hence using these 2 OT calls the parties $P_i$ and $P_j$ secret share $x_i y_j + y_i x_j$. For all $i$, let $z_{ij}$ denote the value $r_1^{ij} + r_2^{ij}$ for all $j > i$,

and let $z_{ij}$ denote the value $o_1^{ij} + o_2^{ij}$ for all $j < i$. Now, each $P_i$ can compute the value $z_i = x_i \cdot y_i + \sum_{j \neq i} z_{ij}$ for $i > 1$ and $z_1 = 1 + x_1 \cdot y_1 + \sum_{j \neq 1} z_{1j}$. These values $\{z_i\}$ form shares of $\mathrm{NAND}(x, y)$. Formally, the security of this $\mathsf{NAND}$ protocol over noiseless channels has been analyzed in prior works [GMW87, Gol04], and we state it as a formal theorem in Appendix G.2.

*Security of $\Pi_{\mathsf{NAND}}$ composed with* $\mathsf{A2A}$. Having described $\Pi_{\mathsf{NAND}}$, we now state the security of the composed protocol $\Pi' = \mathsf{A2A}(\Pi_{\mathsf{NAND}})$, which will be executed within each chunk of our IC scheme. Since $\Pi_{\mathsf{NAND}}$ is channel-transparent, simulated transcripts of $\Pi_{\mathsf{NAND}}$ over noiseless channels can also be used to simulate the $\mathsf{A2A}$ transcripts in $\Pi'$. Moreover, the outputs of $\Pi_{\mathsf{NAND}}$ and $\Pi'$ match in the noiseless setting. This allows us to lift the simulation-based security of $\Pi_{\mathsf{NAND}}$ (Theorem 10) to $\Pi'$ over noiseless channels, as formalized next.

**Theorem 7.** *Let $\Pi' = \mathsf{A2A}(\Pi_{\mathsf{NAND}})$, then there exists a simulator $\mathsf{Sim}_{\mathsf{NAND}}$, such that for any set of corrupt parties $\mathcal{A} \subset [n]$ and any inputs $\{x_i, y_i \in \mathbb{Z}_2\}_{i \in [n]}$ the following holds:*

$$(\mathsf{Sim}_{\mathsf{NAND}}(\mathcal{A}; \{x_i, y_i, z_i\}_{i \in \mathcal{A}}), \{z_i\}_{i \in [n]}) \equiv (\mathsf{View}^*_{\Pi'}(\{x_i, y_i\}_{i \in [n]}), \mathsf{Output}_{\Pi'}(\{x_i, y_i\}_{i \in [n]}))$$

*where*

$$\text{For } i \in [n] : z_i \leftarrow_{\$} \{0,1\} \text{ conditioned on } \quad \sum_{i=1}^{n} z_i = 1 + \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right).$$

*GMW* $\mathsf{NAND}$ *over noisy channels.* When executed over noisy $\mathsf{BSC}$ channels, the protocol $\Pi' = \mathsf{A2A}(\Pi_{\mathsf{NAND}})$ might give incorrect output due to channel noise that was not corrected by the $\mathsf{A2A}$ decoding. However, even if the protocol's output is incorrect, it still satisfies a notion of privacy: the augmented view of any set of corrupt parties can be simulated over noisy $\mathsf{BSC}$ channels given only their inputs. We formalize this observation in the following theorem:

**Protocol $\Pi_{\mathsf{NAND}}$**

- **Inputs:** For $i \in [n]$, Party $P_i$ inputs $x_i \in \{0,1\}$ and $y_i \in \{0,1\}$
- **NAND Gate Protocol**
  1. Each pair of parties executes $n(n-1)/2$ instances of $\Pi_{\mathsf{OT}}$ (Figure 1) in parallel, twice, in the following manner. Party $P_i$ does the following.
     - For each $P_j$ with $j > i$:
       (a) Party $P_i$ takes the role of the sender in the first $\Pi_{\mathsf{OT}}$ and sets its input to $(r_1^{ij}, y_i + r_1^{ij})$, where $r_1^{ij}$ is a random bit.
       (b) Party $P_i$ takes the role of the sender in the second $\Pi_{\mathsf{OT}}$ and sets its input to $(r_2^{ij}, x_i + r_2^{ij})$, where $r_2^{ij}$ is a random bit.
     - For each $P_j$ with $j < i$:
       (a) Party $P_i$ takes the role of the receiver in the first $\Pi_{\mathsf{OT}}$ and sets $x_i$ as its input. Let $P_i$'s output be $o_1^{ij}$.
       (b) Party $P_i$ takes the role of the receiver in the second $\Pi_{\mathsf{OT}}$ and sets $y_i$ as its input. Let $P_i$'s output be $o_2^{ij}$.
  2. Party $P_i$ sets $z_{ij} = r_1^{ij} + r_2^{ij}$ for $j > i$ and $z_{ij} = o_1^{ij} + o_2^{ij}$ for $j < i$.
- **Output Computation.** Each $P_i$ computes: $z_i \leftarrow x_i y_i + \sum_{j \neq i} z_{ij} + \mathbf{1}_{\{i \stackrel{?}{=} 1\}}$. These values form shares of $1 + x \cdot y$.

Fig. 2: Protocol $\Pi_{\mathsf{NAND}}$

**Theorem 8.** *For any set of corrupt parties $\mathcal{A} \subset [n]$, the $n$-party* NAND *protocol $\Pi' = \mathsf{A2A}(\Pi_{\mathsf{NAND}})$ over* BSC *channel satisfies the transparent-channel privacy property: there exist a simulator $\mathsf{Sim}^{\mathsf{BSC}}_{\mathsf{NAND}}$ such that for any inputs $\{x_i, y_i \in \mathbb{Z}_2\}_{i\in[n]}$ the following holds:*

$$\mathsf{Sim}^{\mathsf{BSC}}_{\mathsf{NAND}}(\mathcal{A}; \{x_i, y_i\}_{i\in\mathcal{A}}) \equiv \mathsf{View}^{\mathsf{BSC},*}_{\Pi'}(\{x_i, y_i\}_{i\in[n]})$$

**Simulator** $\mathsf{Sim}^{\mathsf{BSC}}_{\mathsf{NAND}}(\mathcal{A}, x_{\mathcal{A}}, y_{\mathcal{A}})$

1. For $p \in \{1, \ldots, 6\}$: (for each round of NAND)
   - Simulate A2A protocol over $\mathsf{BSC}_\varepsilon$ channels where each corrupt party picks their A2A input as specified by $p^{th}$ round in the $\mathsf{A2A}(\Pi_{\mathsf{NAND}})$ protocol and each honest party picks a random bit
2. Output simulated augmented corrupted party views

Fig. 3: Simulator $\mathsf{Sim}^{\mathsf{BSC}}_{\mathsf{NAND}}$

The simulator $\mathsf{Sim}^{\mathsf{BSC}}_{\mathsf{NAND}}$ (Figure 3) is a randomized algorithm that internally samples BSC noise and honest parties' randomness. In the full MPC simulator (Figure 7), we will want to sample the channel noise and explicitly input it to the NAND simulator. We therefore introduce the extended notation $\mathsf{Sim}^{\mathsf{BSC}}_{\mathsf{NAND}}(\mathcal{A}; \{x_i, y_i\}_{i\in\mathcal{A}}; r)$, where $r \in \{0,1\}^T$ is a bit string specifying the channel noise, with $T$ equal to the total number of bits communicated in $\mathsf{A2A}(\Pi_{\mathsf{NAND}})$. Each component $r_i$ is drawn independently from the Bernoulli distribution $\mathcal{N}(\varepsilon)$ (i.e., $r_i = 1$ with probability $\varepsilon$, representing a bit flip on the channel). When $r = 0^T$, no channel noise occurs, and the protocol reduces to the noiseless setting.

### 4.2 Security Proof for $\Pi^{\mathsf{BSC}}_{\mathsf{MPC}}$

We now outline the security proof for protocol $\Pi^{\mathsf{BSC}}_{\mathsf{MPC}}$; the complete simulator description and all proofs appear in Appendix G.4.

*Simulation strategy.* The key challenge is that the IC scheme may cause parties to rewind and re-execute the NAND sub-protocol for a given gate multiple times before the protocol completes. Each re-execution uses fresh correlated randomness. Our simulator distinguishes between two types of NAND sub-protocol executions:

1. **Non-output executions:** These are intermediate or output NAND gates executions that are later rewound due to detected inconsistencies. For these, we use the noisy simulator $\mathsf{Sim}^{\mathsf{BSC}}_{\mathsf{NAND}}$ (Theorem 8), which simulates the view using only the corrupt parties' inputs to that gate.
2. **Final executions of output gates:** These executions of output gates are not subsequently rewound. For these, we use the noiseless simulator $\mathsf{Sim}_{\mathsf{NAND}}$ (Theorem 7), which takes as additional input the shares of the gate's output consistent with the ideal functionality's output.

For non-output gates and rewound executions, the simulator need not produce shares consistent with any particular output value (since these shares are either discarded or do not directly determine the final output). For final executions of output gates, however, the simulated shares must be consistent with the ideal output that will be reconstructed.

The formal privacy proof presented in Appendix G.4 proceeds by induction over the number of evaluated chunks in the protocol, where each chunk involves a $\mathsf{NAND}$ protocol execution. The base case establishes that the simulated and real views are perfectly indistinguishable after the initialization phase. For the inductive step, we show that if the views are statistically close after $k-1$ chunks, they remain close after chunk $k$. This follows from two observations: (i) each chunk uses independent $\mathsf{ROT}$ correlations, so the $\mathsf{A2A\text{-}NAND}$ view in the $k$-th chunk is independent of previous chunks; and (ii) the $\mathsf{A2A\ NAND}$ view is perfectly simulable by either $\mathsf{Sim}_{\mathsf{NAND}}^{\mathsf{BSC}}$ or $\mathsf{Sim}_{\mathsf{NAND}}$, depending on the case. Combining the evaluation-phase indistinguishability (Lemma 15) with the reconstruction phase yields the full security statement (Theorem 11), with overall statistical distance at most $2^{-\Omega(\kappa)}$.

## References


ABE+19. Noga Alon, Mark Braverman, Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Reliable communication over highly connected noisy networks. *Distributed Computing*, 32(6):505–515, 2019.

Bea91. Donald Beaver. Foundations of secure interactive computing. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 377–391. Springer, 1991.

BEGH17. Mark Braverman, Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Constant-rate coding for multiparty interactive communication is impossible. *J. ACM*, 65(1):4:1–4:41, December 2017.

BGMO17. Mark Braverman, Ran Gelles, Jieming Mao, and Rafail Ostrovsky. Coding for interactive communication correcting insertions and deletions. *IEEE Transactions on Information Theory*, 63(10):6256–6270, Oct 2017.

BGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM, 1988.

Bra12. Mark Braverman. Towards deterministic tree code constructions. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 161–167, New York, NY, USA, 2012. Association for Computing Machinery.

Can00. Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, January 2000.

CCD88. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 11–19. ACM, 1988.

CDLR16. Ignacio Cascudo, Ivan Damgård, Felipe Lacerda, and Samuel Ranellucci. Oblivious transfer from any non-trivial elastic noisy channel via secret key agreement. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part I*, volume 9985 of *Lecture Notes in Computer Science*, pages 204–234, Beijing, China, October 31 – November 3, 2016. Springer Berlin Heidelberg, Germany.

CGH19. Keren Censor-Hillel, Ran Gelles, and Bernhard Haeupler. Making asynchronous distributed computations robust to noise. *Distributed Computing*, 32(5):405–421, Oct 2019.

CK88. C. Crepeau and J. Kilian. Achieving oblivious transfer using weakened security assumptions. In *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, pages 42–52, 1988.

CLPS20. T-H. Hubert Chan, Zhibin Liang, Antigoni Polychroniadou, and Elaine Shi. Small memory robust simulation of client-server interactive protocols over oblivious noisy channels. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, pages 2349–2365, 2020.

CPT13. Kai-Min Chung, Rafael Pass, and Sidharth Telang. Knowledge-preserving interactive coding. In *Proceedings of the 54th annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 449–458, 2013.

CS20. Gil Cohen and Shahar Samocha. Palette-Alternating Tree Codes. In Shubhangi Saraf, editor, *35th Computational Complexity Conference (CCC 2020)*, volume 169 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:29, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

DFMS04. Ivan Damgård, Serge Fehr, Kirill Morozov, and Louis Salvail. Unfair noisy channels and oblivious transfer. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 355–373, Cambridge, MA, USA, February 19–21, 2004. Springer Berlin Heidelberg, Germany.

DN03. Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 247–264. Springer, 2003.

EGL82. Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology – CRYPTO'82*, pages 205–210, Santa Barbara, CA, USA, 1982. Plenum Press, New York, USA.

Gel17. Ran Gelles. Coding for interactive communication: A survey. *Foundations and Trends® in Theoretical Computer Science*, 13(1–2):1–157, 2017.

GI05. V. Guruswami and P. Indyk. Linear-time encodable/decodable codes with near-optimal rate. *IEEE Trans. on Information Theory*, 51(10):3393–3400, 2005.

GK19. Ran Gelles and Yael T. Kalai. Constant-rate interactive coding is impossible, even in constant-degree networks. *IEEE Transactions on Information Theory*, 65(6):3812–3829, 2019.

GKR21. Ran Gelles, Yael Tauman Kalai, and Govind Ramnarayan. Efficient multiparty interactive coding—part I: Oblivious insertions, deletions and substitutions. *IEEE Transactions on Information Theory*, 67(6):3411–3437, 2021.

GKR22. Ran Gelles, Yael Tauman Kalai, and Govind Ramnarayan. Efficient multiparty interactive coding—part II: Non-oblivious noise. *IEEE Transactions on Information Theory*, 68(7):4723–4749, 2022.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987.

Gol04. Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

GPZ18. Ran Gelles, Anat Paskin-Cherniavsky, and Vassilis Zikas. Secure two-party computation over unreliable channels. In *Security and Cryptography for Networks: 11th International Conference, SCN 2018, Amalfi, Italy, September 5–7, 2018, Proceedings*, page 445–463, Berlin, Heidelberg, 2018. Springer-Verlag.

GSW15. Ran Gelles, Amit Sahai, and Akshay Wadia. Private interactive communication across an adversarial channel. *IEEE Transactions on Information Theory*, 61(12):6860–6875, 2015.

Hae14. Bernhard Haeupler. Interactive channel capacity revisited. In *55th Annual Symposium on Foundations of Computer Science*, pages 226–235, Philadelphia, PA, USA, October 18–21, 2014. IEEE Computer Society Press.

HIKN08. Danny Harnik, Yuval Ishai, Eyal Kushilevitz, and Jesper Buus Nielsen. Ot-combiners via secure computation. In Ran Canetti, editor, *Theory of Cryptography*, pages 393–411, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

HS16. William M. Hoza and Leonard J. Schulman. The adversarial noise threshold for distributed protocols. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 240–258, 2016.

IKO+11. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, Amit Sahai, and Jürg Wullschleger. Constant-rate oblivious transfer from noisy channels. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 667–684, Santa Barbara, CA, USA, August 14–18, 2011. Springer Berlin Heidelberg, Germany.

IPS08. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 572–591, Santa Barbara, CA, USA, August 17–21, 2008. Springer Berlin Heidelberg, Germany.

JKL15. 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.

KMS16. Dakshita Khurana, Hemanta K. Maji, and Amit Sahai. Secure computation from elastic noisy channels. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 184–212, Vienna, Austria, May 8–12, 2016. Springer Berlin Heidelberg, Germany.

LV15. Allison Lewko and Ellen Vitercik. Balancing communication for multi-party interactive coding, 2015.

MG24. Manuj Mukherjee and Ran Gelles. Computation in server-assisted noisy networks. In *2024 IEEE International Symposium on Information Theory (ISIT)*, pages 3297–3301, 2024.

Rab81. M. O. Rabin. How to exchange secrets with oblivious transfer. Technical Report TR-81, Aiken Computation Lab, Harvard University, 1981.

Rot06. Ron Roth. *Introduction to Coding Theory*. Cambridge University Press, 2006.

RS94. 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, 1994.

Sch92. Leonard J. Schulman. Communication on noisy channels: a coding theorem for computation. *Foundations of Computer Science, Annual IEEE Symposium on*, pages 724–733, 1992.

Sch93. 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.

Sha48. 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.

Spi95. Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, STOC '95, pages 388–397, New York, NY, USA, 1995. ACM.

Wul09. Jürg Wullschleger. Oblivious transfer from weak noisy channels. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 332–349. Springer Berlin Heidelberg, Germany, March 15–17, 2009.

Yao82. Andrew C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 160–164, 1982.

## A Rewinding GMW potentially harms privacy

In this section we give the detail about *Issue I* discussed in Section 1.1 and show a specific situation where rewinding the GMW protocol in the middle of a NAND computation violates its privacy.

To illustrate this issue, consider the simpler two-party AND functionality, where Alice and Bob hold input bits $x$ and $y$, respectively, and the protocol outputs random XOR shares of $x \wedge y$ to both parties. Figure 4 presents a protocol for realizing this functionality in the random OT hybrid model [Bea91]. In this protocol, the parties effectively implement a 1-out-of-2 OT by exchanging three-bit messages. Alice's OT input messages are $(r, r+x)$, where $r$ is a uniformly random bit and $+$ denotes XOR. Bob's OT input is the selection bit $y$, and he receives the corresponding message $r' = r + (x \wedge y)$ as output. Hence, the pair $(r, r')$ forms an XOR sharing of $x \wedge y$. This functionality is realized via the random OT primitive, which provides correlated OT triples: Alice receives random bits $(r_0, r_1)$, and Bob receives a random selection bit $b$ along with the corresponding message $r_b$ from Alice's pair.

**Protocol $\Pi_{\mathsf{AND}}$**

**Inputs:** Alice input is a bit $x$; Bob inputs a bit $y$.

1. Parties make a call to $\mathcal{F}_{\mathsf{ROT}}$: Alice gets $r_0, r_1$, while Bob gets $b$ and $r_b$ as output.
2. Bob sends $c = y + b$ to Alice, and let its received bit be $c'$.
3. Alice samples $r \leftarrow_{\$} \{0,1\}$ and let $(m_0, m_1) = (r, r + x)$.
4. Alice sends $m_0 + r_{c'}, m_1 + r_{1-c'}$ to Bob, and let the received bits be $(z_0, z_1)$.
5. Alice outputs $r$ and Bob outputs $z_y + r_b$.

Fig. 4: The $\Pi_{\mathsf{AND}}$ Protocol

While this protocol is correct and secure over noiseless channels, we highlight how executing this protocol multiple times with same inputs and randomness (as needed due to rewinding in IC schemes) could lead to privacy violations over $\mathsf{BSC}$ channels. Define $(\varepsilon_1, \varepsilon_2, \varepsilon_3) \in \{0,1\}^3$ as indicators for the event of a bit flip in each of the three bits exchanged in the protocol over a noisy network, respectively. Let say the AND protocol is executed twice with the same inputs $(x, y)$ and and same random OT correlation $((r_0.r_1), (b, r_b))$ but with different noise indicators, say, $(0,0,0)$ and $(1,0,0)$. In these two executions, let the messages received by Bob in the protocol be $(z_0, z_1)$ and $(z'_0, z'_1)$ respectively. Bob can further compute the following given that it has $r_b$:

$$z_y + r_b = (m_y + r_b) + r_b = m_y = r + (x \wedge y)$$
$$z'_{y+1} + r_b = (m_{y+1} + r_b) = m_{y+1} = m_y + x$$

In other words, Bob learns both messages $(m_0, m_1) = (r, r + x)$, one from each execution of the OT functionality. This follows because randomness $r_b$ is used as the mask for Alice's messages $m_y$ and $m_{y+1}$ in the first and second OT invocations respectively. Furthermore, Bob can compute Alice's input as $(z_y + z'_{y+1}) = x$, violating the privacy of the AND functionality. A similar issue arises when executing a single NAND operation in an IC scheme compilation of GMW protocol that allows for rewinding.

## B Omitted Preliminaries

**Definition 2 (Statistical Distance).** *We define $SD(U, V)$ as the statistical distance between distributions $U$ and $V$.*

$$SD(U, V) = \frac{1}{2} \sum_{\omega} |U(\omega) - V(\omega)|.$$

*For two distributions $U$ and $V$ we say that they are $\varepsilon$-*close*, and denote $U \approx_\varepsilon V$ if $SD(U, V) \leqslant \varepsilon$.*

The notion of $\varepsilon$-closeness is easily extendable to probability ensembles (i.e., families of distributions) $U = \{U_\kappa\}_\kappa$ and $V = \{V_\kappa\}_\kappa$ parameterized by a security parameter $\kappa$.

**Definition 3 (Protocol with Correlations).** *An $n$-party protocol $\pi$ with $\mathbb{C}$ correlations for evaluating the functionality $f$ in $|\pi|$ rounds includes a set of next message functions $\pi = (\pi_1, \ldots, \pi_n)$ and a correlation index $t$. At the start of each round $r$, party $P_i$ executes the next message function $\pi_i$ that takes as input some $x_i$, correlation instances $\mathsf{corr}^i[: r/t]$ and the set $m_j^r$ for all $j \neq i$ which denote the set of messages received by party $i$ from party $j$ up to $r$ rounds and computes the next message to be sent to each party. The output of the next message function in the $|\pi|$ round represents the output of the corresponding parties. We denote $\pi_i(x_i, \mathsf{corr}^i)$ to represent output of party $P_i$.*

*We say protocol $\pi$ is **correct**, if for any inputs $x_1, x_2, \ldots, x_n$ and for any correlation $\mathsf{corr} \in \mathbb{C}^{|\pi|/t}$, it holds $(\pi_1(x_1, \mathsf{corr}^1), \ldots, \pi_n(x_n, \mathsf{corr}^n)) = f(x_1, x_2, \ldots, x_n)$*

### B.1 Interactive Coding: Further notation and Preliminaries

For an event $A$, we let $\mathbf{1}_A$ be the indicator variable that takes the value 1 when the event $A$ occurs or otherwise takes the value 0. For $X$, a variable of our algorithm, we denote with $X(r)$ the value of $X$ at the *end* of round $r$ of the algorithm.

**Error-Correcting Codes** We will use the following seminal result by Shannon.

**Lemma 1 (Shannon Coding Theorem [Sha48]).** *For any discrete memoryless channel $\mathsf{Ch}$ with capacity $\mathcal{C}$ and any $k$, there exists a code $\mathsf{ECC} : \{0,1\}^k \to \{0,1\}^n$ and $\mathsf{ECC}^{-1} : \{0,1\}^n \to \{0,1\}^k$ with $n = O(\frac{1}{\mathcal{C}}k)$ such that for any $m \in \{0,1\}^k$ it holds that,*

$$\Pr\left[\mathsf{ECC}^{-1}(\mathsf{Ch}(\mathsf{ECC}(m))) \neq m\right] < 2^{-\Omega(n)}.$$

Recall that the capacity of the $\mathsf{BSC}_\varepsilon$ channel is

$$\mathcal{C}_{\mathsf{BSC}_\varepsilon} = 1 + \varepsilon \log \varepsilon + (1 - \varepsilon)\log(1 - \varepsilon) = O_\varepsilon(1)$$

and that efficient constructions of such codes are well known (e.g., [Spi95, GI05]).

**Outline of RS Scheme** We outline the multiparty interactive coding scheme presented in [RS94]. Each party $i$ maintains a list of messages sent to party $j$ for all $j \neq i$. In each iteration of the scheme, party $i$ decodes the messages sent by each party $j \neq i$. The messages are symbols sent by party $j$ to party $i$ encoded using a tree code. The messages are either the next bit to be sent according to the underlying protocol or a special $\leftarrow$ symbol. After decoding the messages, party $i$ parses the list of messages received from each party $j$ by removing each $\leftarrow$ symbol and the previous undeleted

message from the $\leftarrow$. Then, party $i$ checks if the messages sent in the previous rounds are consistent with those received. Note that due to noise, the tree code decodings may be incorrect, which might result in incorrect messages sent by party $i$ or might have inadvertently deleted correctly received messages. If this consistency check is passed, party $i$ sends the next bit to each party $j$ via the tree code; otherwise, it encodes the $\leftarrow$ symbol with the tree code and sends it to each party.

The tree code symbols are again encoded using an $\mathsf{ECC}$ length $O(d+1)$ where $d$ is the maximum degree of the vertices in the communication graph. This leads to a rate of $O_\varepsilon\left(\frac{1}{d+1}\right)$. To achieve a constant rate, Alon et al. use a so-called all-to-all (A2A) protocol that performs all-to-all communication among $n$ parties in a constant number of rounds with error probability $2^{-\Omega_\varepsilon(\sqrt{n})}$.

**A2A Protocol** The $\mathsf{A2A}$ protocol from [ABE+19] solves the following task. Each party $P_i$ has a vector $b_i = \{b_{i1}, \ldots, b_{in}\}$ of $n$ bits where it wants to send $b_{ij}$ to party $j$. The output of party $P_i$ is a vector $c_i = \{b_{1i}, \ldots, b_{ni}\}$.

The input of the protocol can be represented as a matrix $A$ of size $n \times n$. The contents of the matrix are $A[i][j] = b_{ij}$, where party $i$ wants to send $b_{ij}$ to party $j$. At the start of the protocol, party $i$ knows the bits in the $i$-th row. At the end of the protocol, party $i$ knows the bits of the $i$-th column.

The basic idea of the protocol is as follows. The matrix $A$ can be divided into $n$ submatrices $B_{t,l}$ of size $k \times k$ where $k = \sqrt{n}$. Each $B_{t,l}$ is assigned to a party $(t-1)k + l$. The communication happens in two phases. In the first phase of communication, each party $i$ takes the $k$ bits of $B_{t,l}$ that it knows, encodes them with an $\mathsf{ECC}$ to get $k' = O_\varepsilon(k)$ bits, splits the $k'$ bits into chunks of $k$ bits and sends the $j^{th}$ chunk to party $(ki + j) \mod n$, which relays the chunk to party $(t-1)k + l$. At the end of the first communication phase, each party knows the bits of its submatrix. In the second phase, the transpose direction happens. Each party $(t-1)k + l$ takes the $k$ bits of $B_{t,l}$ that belong to party $i$, encodes them with $\mathsf{ECC}$ to get $k'$ bits, splits the $k'$ bits into chunks of $k$ bits and sends the $j^{th}$ chunk to party $(i + kj) \mod n$ who relays the chunk to party $i$. At the end of the second phase, each party knows the bits it is supposed to learn. The $\mathsf{ECC}$ fails with probability $1 - 2^{-\Omega(\sqrt{n})}$.

The probability of failure of the $\mathsf{A2A}$ protocol is $1 - 2^{-\Omega(\sqrt{n})}$, which is based on the failure probability of $\mathsf{ECC}$ and the $\mathsf{BSC}$ channel error probability.

**Algorithm** A2A

**A Resilient all-to-all (A2A) Communication [ABE+19]**

- **Inputs:** party $i$ for $i \in [n]$ has a single bit $b_{i,j}$ to communicate to $j$ for all $j \in [n]$.
- **Outputs:** party $i$ outputs all the bits address to it, $(b_{1,i}, \ldots, b_{n,i})$.
- **Assume:** error correcting code $\mathsf{ECC} : \{0,1\}^k \to \{0,1\}^{k'}$ with $k' = O(k)$ and constant distance (see Lemma 1)
- **Algorithm:**
  1. Write the $n^2$ bits $\{b_{i,j}\}$ as the matrix $A \in \{0,1\}^{n \times n}$. Set $k = \sqrt{n}$.

2. Re-write $A$ as $n$ disjoint sub-matrices $B_{t,l} \in \{0,1\}^{k\times k}$,

$$A = \begin{pmatrix} B_{1,1} & B_{1,2} & \cdots & B_{1,k} \\ B_{2,1} & B_{2,2} & \cdots & B_{2,k} \\ & & \ddots & \\ B_{k,1} & B_{k,2} & \cdots & B_{k,k} \end{pmatrix},$$

with

$$B_{t,l} = \begin{pmatrix} a_{(t-1)k+1,(l-1)k+1} & a_{(t-1)k+1,(l-1)k+2} & \cdots & a_{(t-1)k+1,lk} \\ a_{(t-1)k+2,(l-1)k+1} & a_{(t-1)k+2,(l-1)k+2} & \cdots & a_{(t-1)k+2,lk} \\ \vdots & & \ddots & \vdots \\ a_{tk,(l-1)k+1} & a_{tk,(l-1)k+2} & \cdots & a_{tk,lk} \end{pmatrix}.$$

3. Associate each $B_{t,l}$ with a unique party, say, $k(t-1)+l$.
4. For any $t, l$, party $i$ encodes the $k$ bits that belong to $B_{t,l}$ with an $\mathsf{ECC}$ to obtain the $k'$ bits $\mathbf{a} = a_1, \ldots, a_{k'}$.
5. Party $i$ splits $\mathbf{a}$ into $k$ disjoint parts of size $O(1)$ bits each and sends the $j$-th part to party $(ki+j) \mod n$ in $O(1)$ rounds. Then, each party relays the message to party $k(t-1)+l$ in $O(1)$ rounds.
6. For any $t, l$, party $k(t-1)+l$ encodes the $i^{th}$ column of $B_{t,l}$ that is to be sent to party $(l-1)k+i$ with an $\mathsf{ECC}$ to obtain the $k'$ bits $\mathbf{d} = d_1, \ldots, d_{k'}$.
7. Party $k(t-1)+l$ splits $\mathbf{d}$ into $k$ disjoint parts of size $O(1)$ bits each and sends the $j$-th part to party $(i+kj) \mod n$ in $O(1)$ rounds. Then, each party relays the message to party $(l-1)k+i$ in $O(1)$ rounds.

Fig. 5: A Resilient all-to-all (A2A) Communication

**Tree Codes** A *tree code* [Sch93] is an encoding technique for strings of $d$-sized alphabets. Informally, a $d$-ary tree code $T$ is an infinite $d$-ary tree, where each edge of the tree is labeled with an alphabet $\Gamma$, and each node has $d$ children. We say that the tree code has distance $\delta$ if, for any two different paths of length $h$ (representing encodings of two strings of equal length), the paths differ by at least $\delta h$ edges. Formally,

**Definition 4 (Tree Codes).** *A tree code with input alphabet $S$, output alphabet $\Sigma$, and distance $\alpha$ is an online encoding algorithm $\mathcal{T} : S^* \to \Sigma^*$ that satisfies the following:*

1. *(online) For any $s \in S^*$, $s = s[1]s[2]\ldots s[k]$, it holds that $\mathcal{T}(s[1]s[2]\ldots s[k]) = \mathcal{T}(s[1]s[2]\ldots s[k-1]) \circ \gamma_k$.*
2. *(distance) For any $s, h_1, h_2 \in S^*$ such that $h_1[1] \neq h_2[1]$ and $|h_1| = |h_2|$, it holds that $d(\mathcal{T}(s \circ h_1), \mathcal{T}(s \circ h_2)) > \alpha|h_1|$,*

*where $d(\cdot,\cdot)$ indicate the Hamming distance between two strings.*

Tree codes of constant distance (and constant output alphabet with respect to the input alphabet and the distance) are known to exists, see [Sch93, Bra12, BGMO17, CS20]; see also [Gel17].

**Lemma 2 ([Sch93]).** *For any $\alpha < 1$ and input alphabet $S$, there exists a tree code of infinite depth and distance $\alpha$ whose output alphabet is of size $|\Sigma| = O(|S|)^{1/(1-\alpha)}$.*

Tree codes are decoded using *minimum distance decoding*. More precisely, the decoding function $\mathcal{T}^{-1} : \Sigma^* \to S^*$ outputs the string whose encoding minimizes the hamming distance to the input,

$$\mathcal{T}^{-1}(r) \triangleq \text{argmin}_{\ s \in S^{|r|}} d(r, \mathcal{T}(s)).$$

Suppose a message $s \in S^*$ is tree coded to obtain $\sigma = \mathcal{T}(s)$, and then sent over some channel (say $\mathsf{BSC}_\varepsilon$ after converting the symbols from $\Sigma$ into $\log|\Sigma|$ bits). Due to channel corruptions, referred to as *channel errors*, some $\sigma' \in \Sigma^*$ is received at the output of the channel. We define $d(\sigma, \sigma')$ to be the *number* of channel errors. We say that a *tree code decoding error* occurs if $s' := \mathcal{T}^{-1}(\sigma') \neq s$. The *length* of the tree code error is defined as the length of the longest suffix in error. More precisely, the length of the tree code error is said to be $h$ if $s'[1, |s|-h] = s[1, |s|-h]$ and $s'[|s|-h+1] \neq s[|s|-h+1]$.

The following lemma relates the length of a tree code error with the number of channel errors in the suffix of the tree-coded message.

**Lemma 3 (Lemma 2.4 in [Gel17]).** *Let $\mathcal{T} : S^* \to \Sigma^*$ be a tree code of distance of distance $\alpha$, and let $s \in S^*$. Suppose $\sigma = \mathcal{T}(s)$ is sent over some channel which outputs $\sigma' \in \Sigma^*$. Then a tree code error of length $h$ occurs only if $d(\sigma[|s| - h + 1, |s|], \sigma'[|s| - h + 1, |s|]) \geqslant \frac{h\alpha}{2}$.*

## C Additional Related Work

### C.1 Secure computation over noisy channels.

Ishai et al. [IKO+11] proposed a protocol for two parties to evaluate a circuit over noisy channels with communication complexity $O(|C|) + \mathsf{poly}(\kappa \cdot \mathsf{depth}(C))$. Their protocol works in the malicious setting and achieves statistical security using the IPS compiler [IPS08]. Following that, Gelles et al. [GPZ18] reduced the communication complexity to $O(|C| + \kappa)$ by using an interactive coding scheme in the semi-honest setting. In the presence of malicious adversaries and adversarial channels, an information-theoretic result by Chung et al. [CPT13] showed that there exists no interactive coding scheme with an error rate better than the naive approach of encoding each message with an ECC. Moreover, the authors demonstrated a positive result in the computational setting, highlighting a trade-off between error rate and communication complexity. A similar result was shown for semi-honest adversaries in the presence of adversarial channels by Gelles et al. [GSW15].

Beyond $\mathsf{BSC}$ channels, a line of research has focused on designing MPC protocols over other noisy channels, including *elastic channels* [DFMS04, Wul09, KMS16]. In an elastic channel, an adversarial receiver can increase the reception reliability unbeknown to the sender, but the sender cannot modify the channel's error probability. The work of Khurana et al. [KMS16] and Cascudo et al. [CDLR16] fully characterizes the parameter for elastic channels that enable secure computation. We believe our techniques can also be applied to design secure MPC protocols with linear overhead over elastic channels for some parameter range, and we leave this as future work.

### C.2 Interactive coding schemes [Gel17].

The seminal work of Schulman [Sch92, Sch93] introduced interactive coding schemes and showed the first schemes that overcome random and adversarial noise in the two-parties setting. In particular,

the work [Sch93] defines the notion of tree-codes and proves that tree-codes with good parameters exist. This result was extended by Rajagopalan and Schulman [RS94] to the multi-party setting, showing a scheme with rate $\Omega_\varepsilon((d+1)^{-1})$ where $d$ is the maximal degree of a node in the network. Alon et al. [ABE+19] extended this scheme to obtain a constant rate for a complete graph. The works [BEGH17, GK19, MG24] prove a lower bound on the coding overhead for certain topologies. For the case of adversarial noise, multi-party coding schemes resilient to some fraction of the noise were presented under different noise models and communication assumptions settings [JKL15, LV15, CGH19, CLPS20, GKR21, GKR22].

## D The ChunkRS coding Scheme

In this part, we present an interactive coding scheme for cliques that operates in "chunks" of size $c$ and succeeds with high probability when communication takes place over $\mathsf{BSC}_\varepsilon$ channels. Our starting point is the multiparty IC scheme of [ABE+19], which simulates any noiseless protocol $\pi$ in a round-by-round fashion and succeeds with probability $1 - 2^{-\Omega(|\pi|\sqrt{n})}$. This scheme builds on the multiparty coding scheme of [RS94] and augments it with an additional layer of reliable all-to-all ($\mathsf{A2A}$) communication, which reduces the effective message-corruption probability to the level required by [RS94] while maintaining constant overhead.

Our new scheme, called ChunkRS (Algorithm 5), directly simulates a chunk of $c = O(1)$ consecutive rounds at a time, where the integer $c \geqslant 1$ is a parameter of the simulation that can be chosen arbitrarily. Let $\pi$ be the noiseless protocol we wish to simulate, and assume that any round beyond $|\pi|$ instructs the parties to exchange zeros. Define $k \triangleq \frac{2|\pi|}{c}$ as the number of iterations required to complete the simulation of $\pi$ (analogous to [RS94], which runs for $2|\pi|$ iterations, each simulating a single bit). The ChunkRS scheme then executes $k$ *iterations* (chunks) of the loop in algorithm 5, which we now describe.

Each iteration consists of two phases. In the first phase, the parties interactively exchange $c$ bits, with each exchange carried out via the $\mathsf{A2A}$ scheme as part of the $\mathsf{SimChunk}()$ procedure (see Algorithm 6). The second phase involves exchanging a single *tree-code* symbol, encoded into bits and again transmitted using the $\mathsf{A2A}$ scheme (see algorithm 5 in Algorithm 5). The tree code[8] used in the second phase is $\mathcal{T} : S^* \to \Sigma^*$ with distance $\alpha = \frac{1}{2}$, where the input alphabet is $S \triangleq \{0,1\}^c \cup \{\mathrm{BK}\}$ and $|\Sigma| = O_\alpha(1)$. Intuitively, the $\mathsf{SimChunk}$ procedure simulates a single chunk, after which the tree code validates the simulation and signals whether it should be accepted or rewound: if the tree-code symbol matches the outcome of $\mathsf{SimChunk}$, the simulation proceeds; otherwise, the next tree-code symbol is BK ("back"), instructing the parties to rewind.

*Detailed Description of the chunked RS scheme.* Every party $P_i$ keeps a record of $n-1$ pairs of strings $(T_{i,j}, R_{i,j}) \in (\{0,1\}^c \cup \{\mathrm{BK}\})^*$, for each $P_j$ it communicates with. To ease notations, $P_i$ will also hold similar variables for itself (i.e., $T_{i,i}, R_{i,i}$) assumed to be all zeros and effectively ignored by the algorithm. We can now denote, for any $i \in [n]$, $T_i = (T_{i,1}, \ldots, T_{i,n})$, and $R_i = (R_{i,1}, \ldots, R_{i,n})$. We will use a similar notation for all other variables of the algorithm: $v_{i,j}, u_{i,j}, z_{i,j}, w_{i,j}$, etc.

The strings $T_i$ and $R_i$ record the sent and received transcripts of $P_i$ respectively. The party $P_i$ will also keep a record of all the tree coded symbols it has received so far from the each other party, namely, the symbols received from $P_j$ are kept at $\rho_{i,j}$.

[8] See Appendix B.1 for the definition and description of tree codes.

**Algorithm 5** The ChunkRS scheme (actions taken by party $P_i$)

**input:** A protocol $\pi$ with zero-padding after $|\pi|$ transmissions; An input $x_i$; A tree-code $\mathcal{T} : S^* \to \Sigma^*$ with distance $\alpha = \frac{1}{2}$; A chunk size $c$.

**initialize** $T_i \leftarrow \emptyset, R_i \leftarrow \emptyset$

**for** iteration $l = 1$ **to** $\frac{2|\pi|}{c}$ **do**
  **for** all $j \in [n]$ **do**
    Let $\rho_{i,j}$ be all symbols received from $P_j$ so far
    Set $w_{i,j} = \mathcal{T}^{-1}(\rho_{i,j})$ be the tree-code decoding of the received symbols
    $\tilde{z}_{i,j} \leftarrow \mathsf{ParseChunk}(w_{i,j})$
    $z_{i,j} \leftarrow \mathsf{ParseChunk}(R_{i,j})$
    $\sigma_{i,j} \leftarrow \mathsf{ParseChunk}(T_{i,j})$
  **end for**

  flag $\leftarrow 1$
  **for** all $j \neq i$ **do** ▷ Checking the consistency of the transcript
    **if** $z_{i,j}$ is not a prefix of $\tilde{z}_{i,j}$ **then**
      flag $\leftarrow 0$
    **end if**
  **end for**

  **if** flag $= 0$ **then**
    ▷ Error detected, simulate arbitrary information and send rewind tree-code symbol
    $u_i, v_i \leftarrow \mathsf{SimChunk}(\text{flag}, \tilde{z}_i, z_i, \sigma_i)$
    $T_{i,j} \leftarrow T_{i,j} \circ \text{BK}, \quad \forall j$
    $R_{i,j} \leftarrow R_{i,j} \circ \text{BK}, \quad \forall j$

    Send the $l$-th symbol of $\mathcal{T}(T_{i,j})$ to $P_j$ for all $j \neq i$ via $\mathsf{A2A}$ scheme

  **else** (flag $= 1$)
    ▷ No error detected, simulate next chunk and send the respective tree-code symbol
    $u_i, v_i \leftarrow \mathsf{SimChunk}(\text{flag}, \tilde{z}_i, z_i, \sigma_i)$
    $T_{i,j} \leftarrow T_{i,j} \circ u_{i,j}, \quad \forall j$
    $R_{i,j} \leftarrow R_{i,j} \circ v_{i,j}, \quad \forall j$

    Send the $l$-th symbol of $\mathcal{T}(T_{i,j})$ to $P_j$ for all $j$ via $\mathsf{A2A}$ scheme
  **end if**
**end for**

**return** $\mathsf{ParseChunk}(T_{i,j}), \mathsf{ParseChunk}(R_{i,j})$ truncated to $\frac{|\pi|}{c}$ chunks, for all $j$.

**Algorithm 6** The SimChunk procedure for chunk $l$ (actions taken by party $P_i$)

**input:** flag, $\tilde{z}_i, z_i, \sigma_i$

**initialize:** $u_{i,j}, v_{i,j} \leftarrow \emptyset, \forall j \in [n]$

**if** flag $= 1$ **then**
  **for** $q = 1$ **to** $c$ **do**
    **for** all $j$ **do**
      $u_{i,j}[q] \leftarrow$ Next bit $P_i$ should send to $P_j$ based on $\pi(x_i, z_i \circ v_i)$
    **end for**

    Using A2A, send $u_{i,j}[q]$ to all $P_j$'s and receive their respective bit $temp_{i,j}[q]$.

    **for** all $j$ **do**
      $r \leftarrow |z_{i,j}|$
      **if** $|\tilde{z}_{i,j}| > r$ **then** ▷ Use previously sent bit, if $P_j$ is ahead
        $v_{i,j}[q] \leftarrow$ the $q$-th bit of the string $\tilde{z}_{i,j}[r+1]$
      **else** ▷ $P_i, P_j$ simulate the same chunk, record the last simulated bit
        $v_{i,j} \leftarrow v_{i,j} \circ temp_{i,j}[q]$
      **end if**
    **end for**
  **end for**

**else** ▷ flag $= 0$, send arbitrary information
  Execute A2A for $c$ consecutive times, with the all-zero string as input.
  $u_{i,j} \leftarrow 0^c, \quad \forall j \neq i$
  $v_{i,j} \leftarrow 0^c, \quad \forall j \neq i$
**end if**

**return** $u_i, v_i$

To proceed with the description of ChunkRS, we define a parse operation, ParseChunk(), similar to the parse operation in [RS94], yet we work in chunks of $c$ bits rather than in bits. The parse operation takes as input a string over the alphabet $S$, parses it symbol by symbol consecutively and records each symbol. However, once a BK symbol is encountered, instead of recording it, the procedure deletes the last recorded symbol (if any). If the operation runs out of symbols to delete while encountering a BK symbol, then ParseChunk returns the null string.

We continue describing ChunkRS from the view of party $P_i$. Suppose we are at the beginning of the $l$-th iteration (chunk), for some $l \in [k]$. At first, party $P_i$ decodes the $(l-1)$-length string of tree-code symbols it has received so far from $P_j$ (for each $j \in [n]$), and obtains the decoded string $w_{i,j} \in S^{l-1}$. Next, $P_i$ parses $w_{i,j}$ to obtain $\tilde{z}_{i,j}$, which is the estimated sent messages of $P_j$ for the current round. To verify that the computation so far makes sense, $\tilde{z}_{i,j}$ must be contrasted with the parsed recorded sent and received transcript, $z_{i,j}, \sigma_{i,j}$, generated from $R_{i,j}, T_{i,j}$ respectively (algorithm 5). We comment here that the lengths of all the string $z_{i,j}, \sigma_{i,j}$ are same for all $j$'s, since the BK symbol is always simultaneously added to all $T_{i,j}, R_{i,j}$ for all $j \in [n]$ in Algorithm 5.

Next, party $P_i$ checks whether its actions so far are consistent with the protocol $\pi$. Specifically, it verifies that its recorded transcript $z_{i,j}$ agrees with the prefix of $\tilde{z}_{i,j}$ of the same length (algorithm 5). If this condition holds for all parties $P_j$, the consistency check passes and the flag variable flag is set to 1; otherwise, if the check fails for even a single $P_j$, we set flag $= 0$. We note here that as a byproduct of the above consistency check, Algorithm 5 ensures that $z_i$ and $\sigma_i$ are also always consistent.

Following the consistency check, $P_i$ begins the first phase of communication for chunk $l$ (Algorithm 6). In this first communication phase, $P_i$ simulates the next chunk of $\pi$ by executing the SimChunk procedure. If the consistency check in this chuck failed (algorithm 5), $P_i$ already knows that it needs to delete its incorrect transcript. So, in this phase, it simply sends $c$ successive zeroes to all other parties using the A2A scheme, irrespective of what it received. Otherwise, it will simulate the next chunk according to $\pi$ (algorithm 5). Recall that when more than $\frac{|\pi|}{c}$ chunks have been successfully simulated, we zero pad $\pi$ for any chunk beyond the first $\frac{|\pi|}{c}$.

A key point here is what should $P_i$ consider as the received bits from other parties, in order to know what next bit to send next when simulating the next chunk of $\pi$. Suppose $|z_{i,j}| = |\tilde{z}_{i,j}|$, i.e., $P_i$ has decoded in this iteration exactly the same amount of (parsed) symbols as it has recorded so far. Then, $P_i$ believes everything is synchronized between itself and $P_j$, and assumes that the interaction with $P_j$ (performed in algorithm 6 in Algorithm 6) is the correct continuation of the simulation. However, if the received symbols indicate a longer transcript than recorded, $|z_{i,j}| < |\tilde{z}_{i,j}|$ then $P_i$ will base the simulation on the estimated chunk received from $P_j$, i.e., on $\tilde{z}_{i,j}[|z_{i,j}|+1]$ (algorithm 6 in algorithm 6).

Upon completing the simulation of the current chunk, party $P_i$ records the simulated chunk(s) $u_{i,j}, v_{i,j} \in \{0,1\}^c$, for all $j \neq i$; these are the $c$ bits $P_i$ sent and received from $P_j$, respectively. The detailed simulation of a specific chunk is given in Algorithm 6.

To proceeds with the second phase of communication in iteration (chunk) $l$, party $P_i$ appends BK to $T_{i,j}, R_{i,j}$ for any $P_j$ that did not pass the consistency check (algorithm 5). If the check did pass, then $P_i$ appends $T_{i,j}, R_{i,j}$ respectively with $u_{i,j}, v_{i,j}$ available from the simulation step (algorithm 5). Next, $P_i$ performs the tree code encodings $\mathcal{T}(T_{i,j})$, and wishes to send the last (i.e., the $l$-th) symbol of $\mathcal{T}(T_{i,j})$ to $P_j$. To do so, it encodes that symbol from $\Sigma$ into $\log|\Sigma|$ bits and exchanges them one-by-one using the A2A scheme. This concludes the second phase of communication the $l$-th chunked round.

After running $k = \frac{2|\pi|}{c}$ successive chunked rounds, $P_i$ outputs the parsed versions of $T_{i,j}, R_{i,j}$, for all parties $P_j$, as its respective sent and received transcripts, truncated to $\frac{|\pi|}{c}$ chunks, i.e., to $|\pi|$ bits.

The proposed ChunkRS scheme satisfies the following theorem, which is formally proved in Appendix D.1.

**Theorem 9.** *Consider a clique of $n$ parties communicating over $\mathsf{BSC}_\varepsilon$ channels, for some $\varepsilon < 1/2$. Then, the ChunkRS scheme with parameter $c > 1$ (Algorithm 5) successfully simulates any fully-utilized protocol $\pi$ in chunks of size $c$, using $O_\varepsilon(|\pi|)$ rounds, with error probability $2^{-\Omega_{\varepsilon,c}(|\pi|\sqrt{n})}$.*

### D.1 Analysis of the ChunkRS scheme

Towards proving Theorem 9 we first set up some required notations. The *progress* made by a party in ChunkRS is the number of chunks of $\pi$ it has successfully simulated. More precisely, the progress made by $P_i$ after $t$ iterations (chunks), denoted $\mathrm{RP}(P_i, t)$, is defined as

$$\mathrm{RP}(P_i, t) \triangleq \text{argmax}_{\ l \in [t]} \Big\{ z_i(t)[1, l] = m_{\to i}(l \cdot c) \text{ and } \sigma_i(t)[1, l] = m_{i \to}(l \cdot c) \Big\}, \tag{2}$$

when the equality refers to the respective induced bit-strings. Note that we use $z_i(t)$ and $\sigma_i(t)$ as these correspond to the parsed versions of the transcripts $R_i$ and $T_i$ after they are updated in the $t$-th iteration, treated as strings over the set $S$ (where every symbol is $\{0,1\}^c$, i.e., $c$-bits). Also recall that $m_{i\to}(l \cdot c)$ and $m_{\to i}(l \cdot c)$ are the correct sent and received transcripts of $P_i$ in $\pi$ up to round $l \cdot c$ (of $\pi$), which corresponds to $l$ chunks of $c$ bits. Recalling that $k = \frac{2|\pi|}{c}$, the simulation ChunkRS will be successful if $\mathrm{RP}(P_i, k) \geqslant \frac{k}{2}$, at all parties $P_i$.

To analyze the progress, we define a *historical path*[9] of length $t$, as a sequence of $t$ parties, $\phi \in (\{P_i\}_{i\in[n]})^t$. The sequence may contain the same party multiple times, i.e., it is possible that $\phi[t'] = \phi[t'']$ for any $1 \leqslant t' < t'' \leqslant t$. The set of all historical paths of length $t$ is denoted $\mathcal{H}_t$.

Next we define three more quantities. For any $i \in [n]$, define $\mathrm{B}(P_i, t)$ to be the number of iterations up to and including $t$, where BK symbols were sent by $P_i$. It immediately follows from (2) and the definition of the $\mathsf{ParseChunk}$ operation that

$$\mathrm{RP}(P_i, t) \leqslant t - 2\mathrm{B}(P_i, t). \tag{3}$$

We say that a *$\mathsf{SimChunk}$ decoding error* is made at iteration $l$ if there exists some party $P_i$ which made a decoding error in the $\mathsf{A2A}$ algorithm in one of the $c$ rounds of $\mathsf{SimChunk}$ at iteration $l$. Define $\mathrm{D}(t)$ to be the maximum number of $\mathsf{SimChunk}$ decoding errors across all historical paths in $\mathcal{H}_t$. More precisely,

$$\mathrm{D}(t) \triangleq \max_{\phi \in \mathcal{H}_t} \sum_{l=1}^{t} \mathbf{1}\{\phi(l) \text{ made a } \mathsf{SimChunk} \text{ decoding error at iteration } l\}.$$

We say that a party $P_i$ makes a tree code decoding error in iteration $l$ if it makes a tree code decoding error for the string of symbols sent to it by some party $P_j$, i.e., $T_{j,i} \neq w_{i,j}$.

[9] Here, we divert from the analysis of [RS94], where the historical path is defined for each $P_i$ separately. The reason for this change will become apparent in the analysis of Lemma 4, where the subcase 'propagated error' of [RS94] needs to be broken into two separate sub-cases for our analysis.

For any $\phi \in \mathcal{H}_t$, let $\mathrm{E}(\phi)$ count the instances where party $\phi[l]$ made a tree code decoding error, that is,

$$E(\phi) \triangleq \sum_{l=1}^{|\phi|} \mathbf{1}\{\phi(l) \text{ made a tree code decoding error in iteration } l\}.$$

Also, we say that a tree code error *began* at $\phi[l]$, if the party $\phi[l]$ did not make any tree code decoding error in iteration $l-1$, but it made at least one tree code decoding error in iteration $l$. We say that $\phi \in \mathcal{H}_t$ has *disjoint tree errors* if for any $l$ for which $\phi[l] \neq \phi[l-1]$, the party $\phi[l]$ makes a tree code decoding error at iteration $l$ only if that tree code error began at $\phi[l]$.

### D.2 Proof of Theorem 9

The proof of Theorem 9 follows from the three subsequent lemmas. The first of these lemmas relate progress made by $P_i$ up to iteration $t$ with the number of BKs sent, the number of iterations with decoding errors in SimChunk, and the number of tree code decoding errors.

**Lemma 4.** *Given an execution of Algorithm 5, and for any iteration $t$ and any party $P_i$, there exists a historical path $\phi_{i,t} \in \mathcal{H}_t$ with disjoint tree errors satisfying*

$$t \leqslant RP(P_i, t) + B(P_i, t) + D(t) + E(\phi_{i,t}). \tag{4}$$

As a corollary to Lemma 4, we obtain the following lemma.

**Lemma 5.** *Given an execution of Algorithm 5, and for any iteration $t$ and any party $P_i$, there exists a historical path $\phi_{i,t} \in \mathcal{H}_t$ with disjoint tree errors, such that*

$$E(\phi_{i,t}) + D(t) \geqslant \frac{t - RP(P_i, t)}{2}.$$

*Proof.* Using Lemma 4, there exists $\phi_{i,t} \in \mathcal{H}_t$ with disjoint tree errors, satisfying

$$\begin{aligned} 2t &\leqslant 2\mathrm{RP}(P_i, t) + 2\mathrm{B}(P_i, t) + 2(\mathrm{D}(t) + \mathrm{E}(\phi_{i,t})) \\ &\leqslant \mathrm{RP}(P_i, t) + t + 2(\mathrm{D}(t) + \mathrm{E}(\phi_{i,t})), \end{aligned}$$

where the second inequality follows from Eq. (3). Lemma 5 then follows by rearranging the above. □

The third and final lemma is obtained as a corollary to Lemma 5, and is stated below.

**Lemma 6.** *Following an execution of Algorithm 5, the probability that there exists a party $P_i$ with $RP(P_i, k) < \frac{k}{2}$ is $2^{-\Omega(k\sqrt{n})}$.*

Recalling that ChunkRS leads to successful simulation if $\mathrm{RP}(P_i, k) \geqslant \frac{k}{2}$ for all $i \in [n]$, Lemma 6 ensures that the probability of failure of ChunkRS is $o(1)$. Finally, noting that each iteration of SimChunk involves $O_\varepsilon(c)$ rounds of communication, the round complexity of ChunkRS, which contains $k = O(|\pi|/c)$ iterations, is $O_\varepsilon(|\pi|)$. Therefore, this completes the proof of Theorem 9 modulo the proofs of Lemmas 4 and 6. These are produced in the subsequent subsections.

We begin with the proofs of Lemmas 4 and 6.

**Proof of Lemma 4** To start the proof, we classify the action of $P_i$ in iteration $t$ into six mutually exclusive and exhaustive categories listed below.

- Progress (p): Occurs if $\mathrm{RP}(P_i, t) = \mathrm{RP}(P_i, t-1) + 1$.
- Necessary backup (bk): If BK was sent in iteration $t$ and $\mathrm{RP}(P_i, t) = \mathrm{RP}(P_i, t-1)$.
- Harmful tree error (hte): When neither p or bk occurs, and a tree code decoding error is made by $P_i$.
- Harmful backup (hbk): There was no tree code decoding error, but BK was sent and it deleted a correct chunk, i.e., $\mathrm{RP}(P_i, t) = \mathrm{RP}(P_i, t-1) - 1$.
- Propagated error with a tree code error (pete): There was no tree code decoding error by $P_i$, BK was not sent by $P_i$, a tree code decoding error was made by some other party $P_j, j \neq i$ in iteration $t$, and $\mathrm{RP}(P_i, t-1) = \mathrm{RP}(P_i, t)$.
- Propagated error with SimChunk decoding error (pede): Similar to pete except no party made a tree code decoding error in iteration $t$.

We note here that these categories are similar to those in the progress proof of [RS94], except for pede and pete, which were merged into a single category in [RS94]. It is exactly in the analysis of these categories we see the effect of the chunked simulation, where errors due to the free simulation of a chunk in the SimChunk procedure (Algorithm 6) need to be accounted for.

We shall prove the result by induction. Furthermore, we shall assume in our induction hypothesis that for any $l \leqslant t-1$, and any party $P_i$, the historical path $\phi_{i,l}$ satisfying the induction hypothesis has $\phi_{i,l}[l] = P_i$ unless $P_i$ had pete at iteration $l$.

To proceed, note that the induction hypothesis obviously holds when $t = 0$. We assume that the result holds for all parties upto iteration $t-1$. To complete the proof, we fix a party $P_i$, and show that the hypothesis of Lemma 4 holds for iteration $t$, in a case by case basis.

To begin with, observe that by definition $\mathrm{D}(t-1) \leqslant \mathrm{D}(t)$. Second, note that if $\phi_{i,t} = \phi_{i,t-1} \circ P_i$, then $\mathrm{E}(\phi_{i,t}) \geqslant \mathrm{E}(\phi_{i,t-1})$. Furthermore, we claim that $\phi_{i,t}$ obtained in this fashion will have disjoint tree errors. To see this, observe that by the induction hypothesis, $\phi_{i,t-1}$ has disjoint tree errors. Now, if $\phi_{i,t-1}(t-1) = P_i$, then by default $\phi_{i,t}$ will have disjoint tree errors. On the other hand, if $\phi_{i,t-1}(t-1) \neq P_i$, then by the induction hypothesis, $P_i$ must have pete in iteration $t-1$. Thus, by definition of pete, $P_i$ made no tree code decoding error in $t-1$. Hence, even if $P_i$ made a tree code decoding error at $t$, that tree code error began at $t$. So, $\phi_{i,t}$ will have disjoint tree errors.

We first do away with the easy cases. If p occurred then BK couldn't have been sent by $P_i$ in iteration $t$. Hence $\mathrm{B}(P_i, t-1) = \mathrm{B}(P_i, t)$. Also, define $\phi_{i,t} = \phi_{i,t-1} \circ P_i$. Then, the induction hypothesis holds for $P_i$ at $t$, by using the induction hypothesis at $t-1$ and $\mathrm{RP}(P_i, t) = \mathrm{RP}(P_i, t-1) + 1$. Next, assume bk occurred. Hence $\mathrm{B}(P_i, t) = \mathrm{B}(P_i, t-1) + 1$ and $\mathrm{RP}(P_i, t) = \mathrm{RP}(P_i, t-1)$. Again, define $\phi_{i,t} = \phi_{i,t-1} \circ P_i$. Therefore, the induction hypothesis at $t$ holds by using the hypothesis at $t-1$.

Next, assume hte has occurred, and define $\phi_{i,t} = \phi_{i,t-1} \circ P_i$. Hence, we must have $\mathrm{E}(\phi_{i,t}) = \mathrm{E}(\phi_{i,t-1}) + 1$, as hte implies $P_i$ making a tree code decoding error at $t$. Next, note that among the remaining three quantities in the right hand side of the induction hypothesis, only $\mathrm{RP}(P_i, t)$ is not monotonic in $t$. Now, if $\mathrm{RP}(P_i, t) = \mathrm{RP}(P_i, t-1)$, then the induction hypothesis holds using $\mathrm{E}(\phi_{i,t}) = \mathrm{E}(\phi_{i,t-1}) + 1$. Otherwise, if $\mathrm{RP}(P_i, t) = \mathrm{RP}(P_i, t-1) - 1$, then it must be the case that a BK was sent by $P_i$ in iteration $t$, and hence $\mathrm{B}(P_i, t) = \mathrm{B}(P_i, t-1) + 1$. Thus, in conjunction with $\mathrm{E}(\phi_{i,t}) = \mathrm{E}(\phi_{i,t-1}) + 1$, this proves the induction hypothesis at $t$.

We now move to the three remaining cases which require slightly more involved analysis. Before beginning the analysis, note that if for any $j \neq i$, if $\phi_{i,t} = \phi_{j,t-1} \circ P_i$, then obviously $\mathrm{E}(\phi_{i,t}) \geqslant \mathrm{E}(\phi_{j,t-1})$. Furthermore, since in the remaining cases $P_i$ doesn't make any tree code decoding error

at $t$, and $\phi_{j,t-1}$ has disjoint tree errors by the induction hypothesis, therefore, any $\phi_{i,t}$ obtained in the above fashion also has disjoint tree errors.

To proceed, we consider that hbk has occurred. Hence, we must have $\mathrm{B}(P_i,t) = \mathrm{B}(P_i,t-1)+1$ and $\mathrm{RP}(P_i,t) = \mathrm{RP}(P_i,t-1)-1$. Now, since BK was sent by $P_i$, there must have been a consistency check failure (i.e., flag = 0 in this iteration). We first note that no tree code decoding error was made by $P_i$ at iteration $t$. Hence, for all $j \neq i$, the variable $\tilde{z}_{i,j}(t)$ is exactly the parsed version of the string of symbols sent by $P_j$ to $P_i$ so far.

We now divide the analysis into two subcases. First we assume that there was some $j^* \neq i$ such that $|\tilde{z}_{i,j^*}(t)| < |z_{i,j^*}(t)|$, and this led to the consistency check failure. This means that $\mathrm{B}(P_{j^*},t-1) \geqslant \mathrm{B}(P_i,t-1)+1$. Moreover, since sending a BK symbol in $t$ resulted in $\mathrm{RP}(P_i,t)$ decreasing by 1, we must have $\mathrm{RP}(P_i,t-1) = t-1-2\mathrm{B}(P_i,t-1)$. Thus, using (3), we get

$$\begin{aligned}
\mathrm{RP}(P_{j^*},t-1) &\leqslant t-1-2\mathrm{B}(P_{j^*},t-1) \\
&\overset{(a)}{=} \mathrm{RP}(P_i,t-1)+2\mathrm{B}(P_i,t-1)-2\mathrm{B}(P_{j^*},t-1) \\
&\overset{(b)}{\leqslant} \mathrm{RP}(P_i,t-1)+\mathrm{B}(P_i,t-1)-\mathrm{B}(P_{j^*},t-1)-1, \qquad (5)
\end{aligned}$$

where $(a)$ uses $\mathrm{RP}(P_i,t-1) = t-1-2\mathrm{B}(P_i,t-1)$, and $(b)$ follows from $\mathrm{B}(P_{j^*},t-1) \geqslant \mathrm{B}(P_i,t-1)+1$. Hence, using the induction hypothesis at $P_{j^*}$ at $t-1$, and (5), and by defining $\phi_{i,t} = \phi_{j^*,t-1} \circ P_i$, we have

$$\begin{aligned}
t-1 &\leqslant \mathrm{RP}(P_j,t-1)+\mathrm{B}(P_j,t-1)+\mathrm{D}(t-1)+\mathrm{E}(\phi_{j^*,t-1}) \\
&\leqslant \mathrm{RP}(P_i,t-1)+\mathrm{B}(P_i,t-1)-1+\mathrm{D}(t)+\mathrm{E}(\phi_{i,t}) \\
&= \mathrm{RP}(P_i,t)+\mathrm{B}(P_i,t)-1+\mathrm{D}(t)+\mathrm{E}(t),
\end{aligned}$$

where the last equality follows using $\mathrm{RP}(P_i,t) = \mathrm{RP}(P_i,t-1)-1$ and $\mathrm{B}(P_i,t) = \mathrm{B}(P_i,t-1)+1$. Hence, we have $t \leqslant \mathrm{RP}(P_i,t)+\mathrm{B}(P_i,t)+\mathrm{D}(t)+\mathrm{E}(\phi_{i,t})$, with $\phi_{i,t}$ having disjoint tree errors as required.

Next, assume that the other sub-case of hbk occurred, i.e., for all $j \neq i$, $|\tilde{z}_{i,j}(t)| \geqslant |z_{i,j}(t)|$. This immediately implies that for all $j \neq i$, we must have $\mathrm{B}(P_i,t-1) \geqslant \mathrm{B}(P_j,t-1)$. Next, the failure of consistency check implies that there must be at least one $P_{j^*}$ for which $\tilde{z}_{i,j^*}(t)$ is not consistent with $\pi_0$. This is because $z_{i,j^*}^{(t)}$ must be completely consistent $\pi_0$, as otherwise $\mathrm{RP}(P_i,t-1)$ would not have decreased by 1. Now, since $P_i$ did not commit any tree code decoding error, $\tilde{z}_{i,j}(t)$, for all $j \neq i$, correspond exactly to the parsed version of $P_j$'s sent transcript to $P_i$. Thus, consistency check failure implies that the number of chunks of $\tilde{z}_{i,j^*}(t)$ consistent with $\pi$ is strictly fewer than $|z_{i,j^*}(t)|$. Hence, we have $\mathrm{RP}(P_i,t-1) \geqslant \mathrm{RP}(P_j^*,t-1)+1$. Plugging this and $\mathrm{B}(P_i,t-1) \geqslant \mathrm{B}(P_j^*,t-1)$ in the induction hypothesis for $P_j$ at $t-1$, and defining $\phi_{i,t} = \phi_{j^*,t-1} \circ P_i$, we get

$$\begin{aligned}
t-1 &\leqslant \mathrm{RP}(P_j^*,t-1)+\mathrm{B}(P_j^*,t-1)+\mathrm{D}(t-1)+\mathrm{E}(\phi_{j^*,t-1}) \\
&\leqslant \mathrm{RP}(P_i,t-1)-1+\mathrm{B}(P_i,t-1)+\mathrm{D}(t)+\mathrm{E}(\phi_{i,t}) \\
&= \mathrm{RP}(P_i,t)+\mathrm{B}(P_i,t)-1+\mathrm{D}(t)+\mathrm{E}(\phi_{i,t}),
\end{aligned}$$

where the equality follows using $\mathrm{RP}(P_i,t) = \mathrm{RP}(P_i,t-1)-1$ and $\mathrm{B}(P_i,t) = \mathrm{B}(P_i,t-1)+1$. Hence, we have $t \leqslant \mathrm{RP}(P_i,t)+\mathrm{B}(P_i,t)+\mathrm{D}(t)+\mathrm{E}(\phi_{i,t})$, with $\phi_{i,t}$ having disjoint tree errors as required.

Next, we verify the induction hypothesis if pede occurred. In this case, note that no tree code decoding error by any party for iteration $t$ implies that everyone has the correct version of everyone

else's transcript. In particular, this means $\tilde{z}_{i,j}(t) = \sigma_{j,i}(t)$ for all $j \neq i$. Now, since consistency check was passed, we must have $|\tilde{z}_{i,j}(t)| \geqslant |z_{i,j}(t)|$ for all $j \neq i$. This implies $\mathrm{B}(P_i, t-1) \geqslant \mathrm{B}(P_j, t-1)$ for all $j \neq i$.

Firstly, we prove that $\mathrm{RP}(P_j, t-1) \geqslant \mathrm{RP}(P_i, t-1) + 1$ for all $j \neq i$ cannot occur. Indeed, if $\mathrm{RP}(P_j, t-1) \geqslant \mathrm{RP}(P_i, t-1) + 1$ for all $j \neq i$, then due to the fact that $\tilde{z}_{i,j}(t) = \sigma_{j,i}(t)$ for all $j \neq i$, the SimChunk procedure (Algorithm 6) would ensure that $\mathrm{RP}(P_i, t) = \mathrm{RP}(P_i, t-1) + 1$. We now break the analysis into a pair of subcases. The first subcase assumes that there exists $P_{j^*}$ satisfying $\mathrm{RP}(P_i, t-1) \geqslant \mathrm{RP}(P_{j^*}, t-1) + 1$. Then, using the induction hypothesis at $t-1$ for $P_{j^*}$, and setting $\phi_{i,t} = \phi_{j^*,t-1} \circ P_i$, we get

$$\begin{aligned} t-1 &\leqslant \mathrm{RP}(P_{j^*}, t-1) + \mathrm{B}(P_{j^*}, t-1) + \mathrm{D}(t-1) + \mathrm{E}(\phi_{j^*,t-1}) \\ &\overset{(a)}{\leqslant} \mathrm{RP}(P_i, t-1) - 1 + \mathrm{B}(P_i, t-1) + \mathrm{D}(t) + \mathrm{E}(\phi_{i,t}) \\ &\overset{(b)}{=} \mathrm{RP}(P_i, t) - 1 + \mathrm{B}(P_i, t) + \mathrm{D}(t) + \mathrm{E}(\phi_{i,t}), \end{aligned}$$

where $(a)$ follows by recalling $\mathrm{B}(P_i, t-1) \geqslant \mathrm{B}(P_{j^*}, t-1)$, and $(b)$ uses $\mathrm{RP}(P_i, t-1) = \mathrm{RP}(P_i, t)$ and $\mathrm{B}(P_i, t-1) = \mathrm{B}(P_i, t)$.

If the above subcase doesn't occur, then the set $A = \{P_j : \mathrm{RP}(P_j, t-1) = \mathrm{RP}(P_i, t-1), j \neq i\}$ is non-empty. Since, there was no tree code decoding error at any of the parties, all parties $P_j \in A$ can pass the consistency check only if $\mathrm{RP}(P_j, t-1) = |\sigma_{j,i}(t)|$. Therefore, the only way that $P_i$ could make no progress in this situation would be due to the fact that some party in $A$, or $P_i$ itself, made a decoding error during SimChunk. Let that party be $j^*$, and define $\phi_{i,t} = \phi_{j^*,t-1} \circ P_i$. Now, the decoding error in SimChunk implies $\mathrm{D}(t) = \mathrm{D}(t-1) + 1$. This gives us the induction hypothesis at $t$ via the hypothesis at $t-1$, as well as $\mathrm{RP}(P_i, t) = \mathrm{RP}(P_i, t-1)$ and $\mathrm{B}(P_i, t) = \mathrm{B}(P_i, t-1)$.

Alternatively, it may be the case that some party in $A$ failed the consistency check. Let that party be $P_{j^*}$. Since, every party must have at least as much progress as $P_i$, we must have that $\mathrm{RP}(P_{j^*}, t-1) < t-1-2\mathrm{B}(P_{j^*}, t-1)$. On the other hand, since $P_i$ still passed the consistency check, we still have $\mathrm{RP}(P_i, t-1) = t-1-2\mathrm{B}(P_i, t-1)$. Hence, we must have $\mathrm{B}(P_{j^*}, t-1) \leqslant \mathrm{B}(P_i, t-1) - 1$. Thus, defining $\phi_{i,t} = \phi_{j^*,t-1} \circ P_i$, and using the induction hypothesis at $P_{j^*}$ at iteration $t-1$, we have

$$\begin{aligned} t-1 &\leqslant \mathrm{RP}(P_{j^*}, t-1) + \mathrm{B}(P_{j^*}, t-1) + \mathrm{E}(\phi_{j^*,t-1}) + \mathrm{D}(t-1) \\ &\leqslant \mathrm{RP}(P_i, t-1) + \mathrm{B}(P_i, t-1) - 1 + \mathrm{E}(\phi_{i,t}) + \mathrm{D}(t) \\ &= \mathrm{RP}(P_i, t) + \mathrm{B}(P_i, t) - 1 + \mathrm{E}(\phi_{i,t}) + \mathrm{D}(t), \end{aligned}$$

where the last step follows by recalling $\mathrm{RP}(P_i, t-1) = \mathrm{RP}(P_i, t)$ and $\mathrm{B}(P_i, t-1) = \mathrm{B}(P_i, t)$. This completes the proof of the induction hypothesis when pede occurred for $P_i$ at $t$.

To finish the proof, we need to consider the last case where pete occurs for $P_i$ at $t$. Arguing exactly similar to the pede scenario, we have that $\mathrm{B}(P_j, t-1) \leqslant \mathrm{B}(P_i, t-1)$ and $\mathrm{RP}(P_j, t-1) \geqslant \mathrm{RP}(P_i, t-1)$ for all $j \neq i$. Now, define the sets $A = \{P_j : \mathrm{RP}(P_j, t-1) = \mathrm{RP}(P_i, t-1), j \neq i\}$ and $B = \{P_j : \mathrm{RP}(P_j, t-1) > \mathrm{RP}(P_i, t-1)\}$. Now, we divide the analysis into two subcases.

First, consider that $A = \emptyset$. However, as argued in the pede case, $B \subsetneq [n] \setminus \{i\}$. Hence, there must exist some $j^*$ satisfying $\mathrm{RP}(P_{j^*}, t-1) \leqslant \mathrm{RP}(P_i, t-1) - 1$. Arguing as in the equivalent case for pede and setting $\phi_{i,t} = \phi_{j^*,t-1} \circ P_i$, we obtain the induction hypothesis at $t$. Thus, for the remaining subcases, we assume that $A \neq \emptyset$.

For the second subcase, we assume that there exists $j^* \in A$ that made the tree code decoding error at $t$. Define $\phi_{i,t} = \phi_{j^*,t-1} \circ P_{j^*}$, and observe that $\mathrm{E}(\phi_{i,t}) = \mathrm{E}(\phi_{j^*,t-1}) + 1$. Again, as argued

previously for $\phi_{i,t-1} \circ P_i$, we can argue that $\phi_{j^*,t-1} \circ P_{j^*}$ has disjoint tree errors due to the induction hypothesis. Also, note here that $j^* \neq i$ is being added to get $\phi_{i,t}$, but this is occurring when pete occurred, as required by the induction hypothesis. Now, using the induction hypothesis for $j^*$ at $t-1$, we get

$$\begin{aligned} t-1 &\leqslant \mathrm{RP}(P_{j^*}, t-1) + \mathrm{B}(P_{j^*}, t-1) + \mathrm{D}(t-1) + \mathrm{E}(\phi_{j^*,t-1}) \\ &\overset{(a)}{\leqslant} \mathrm{RP}(P_i, t-1) + \mathrm{B}(P_i, t-1) + \mathrm{D}(t) + \mathrm{E}(\phi_{i,t}) - 1 \\ &\overset{(b)}{=} \mathrm{RP}(P_i, t) + \mathrm{B}(P_i, t) + \mathrm{D}(t) + \mathrm{E}(\phi_{i,t}) - 1 \end{aligned}$$

where $(a)$ follows by recalling $j^* \in A$, $\mathrm{B}(P_i, t-1) \geqslant \mathrm{B}(P_{j^*}, t-1)$, and $\mathrm{E}(\phi_{i,t}) = \mathrm{E}(\phi_{j^*,t-1}) + 1$, and $(b)$ follows by recalling $\mathrm{B}(P_i, t-1) = \mathrm{B}(P_i, t)$ and $\mathrm{RP}(P_i, t-1) = \mathrm{RP}(P_i, t)$ due to pete. Rearranging $(b)$ we obtain the induction hypothesis for $t$.

To finish the proof of Lemma 4, we look at the final subcase where no party in $A$ made a tree code decoding error. Like the pede case before, we can again argue in this case that some party $P_{j^*}$ with $j^* \in A$, or $P_i$ itself must have made a decoding error in SimChunk, or some $P_{j^*}$ with $j^* \in A$ failed the consistency check. In both cases, we define $\phi_{i,t} = \phi_{j^*,t-1} \circ P_i$, and argue exactly similar to the pede case to prove the induction hypothesis.

**Proof of Lemma 6** We begin by fixing a party $P_i$. If $\mathrm{RP}(P_i, k) < \frac{k}{2}$, then, by Lemma 5, there exists a history path $\phi_{i,k}$ with disjoint tree errors satisfying $\mathrm{E}(\phi_{i,k}) + \mathrm{D}(k) > \frac{k}{4}$. We begin by upper bounding the probability of this event.

Now, note that if $\mathrm{E}(\phi_{i,k}) + \mathrm{D}(k) > \frac{k}{4}$, then either $\mathrm{E}(\phi_{i,k}) > \frac{k}{8}$ or $\mathrm{D}(k) > \frac{k}{8}$. We individually bound these probabilities first. To begin with, fix any historical path $\phi \in \mathcal{H}_k$ with disjoint tree errors. For any $l \geqslant 1, h \geqslant 1$, with $l + h - 1 \leqslant k$, let $\phi[l, l+h-1]$ be any successive chunk of tree code errors in $\phi$. By this, we mean that $\phi[l] = \phi[l+1] = \ldots = \phi[l+h-1] = P_j$ for some $j \in [n]$ ($j = i$ is also allowed), and the party $P_j$ made tree code errors in chunked rounds $l, l+1, \ldots, l+h-1$. Since $\phi$ has disjoint tree errors, then $P_j$ made no tree code errors in $l-1$. Also, we either have $\phi[l+h+1] \neq P_j$ or $P_j$ makes no tree code error in iteration $l+h$. Now, since the tree code distance of $\mathcal{T}$ is $\frac{1}{2}$, a tree code error of length $h$ implies at least $\frac{h}{4}$ symbols from the tree code output alphabet $\Sigma$ were wrongly decoded by $P_j$ in the chunked rounds $l$ to $l+h-1$ – see Lemma 3. Therefore, we can conclude that in every historical path $\phi \in \mathcal{H}_k$ with disjoint tree errors, there are at least $\frac{\mathrm{E}(\phi)}{4}$ chunked rounds $l$, where $\phi[l]$ makes a decoding error in the A2A scheme in the second phase of communication.

Now, since $\log |\Sigma| = O(c)$ and each tree code symbol from $\Sigma$ is communicated via $\log |\Sigma|$ rounds of A2A scheme, the probability of channel error, i.e., the probability of decoding a tree code symbol is $q = 2^{-\Omega(\sqrt{n})}$ by Theorem 2. Define the random variable $X_l$ to be the indicator that $\phi[l]$ made a

channel error in iteration $l$. Then $\Pr(X_l = 1) = q$. Thus, we have

$$\begin{aligned}\Pr\left(\mathrm{E}(\phi) > \frac{k}{8}\right) &\leqslant \Pr\left(\sum_{l=1}^{k} X_i \geqslant \frac{k}{32}\right) \\ &= \sum_{s=\frac{k}{32}}^{k} \binom{k}{s} q^s (1-q)^{k-s} \\ &\leqslant 2^k q^{\frac{k}{32}} \\ &= 2^{-\Omega(k\sqrt{n})}, \end{aligned} \tag{6}$$

where the last equality uses $q = 2^{-\Omega(\sqrt{n})}$.

On the other hand, define $Y_l$ to be the indicator random variable of some party making a decoding error during SimChunk for iteration $l$. Since, the communication in SimChunk consists of $c = O(1)$ rounds of A2A, we have by Theorem 2 that $r = \Pr(Y_l = 1) = 2^{-\Omega(\sqrt{n})}$. Therefore, we have

$$\begin{aligned}\Pr\left(\mathrm{D}(k) > \frac{k}{8}\right) &= \Pr\left(\sum_{l=1}^{k} Y_l \geqslant \frac{k}{8}\right) \\ &= \sum_{s=\frac{k}{8}}^{k} \binom{k}{s} r^s (1-r)^{k-s} \\ &\leqslant 2^k r^{\frac{k}{8}} \\ &= 2^{-\Omega(k\sqrt{n})}, \end{aligned} \tag{7}$$

where the last equality again uses $r = 2^{-\Omega(\sqrt{n})}$.

Now, noting that the total number of historical paths in $\mathcal{H}_k$ with disjoint tree errors is upper bounded by $n^k = 2^{k \log n}$, the probability that there exists $\phi_{i,k}$ satisfying $\mathrm{E}(\phi_{i,k}) > \frac{k}{8}$ is upper bounded by $2^{-\Omega(k\sqrt{n})}$ using union bound and (6). This, together with the union bound and (7) upper bounds the probability of the existence of $\phi_{i,k}$ satisfying $\mathrm{E}(\phi_{i,t}) + \mathrm{D}(k) > \frac{k}{4}$ is $2^{-\Omega(\sqrt{n})}$. In other words, $\Pr(\mathrm{RP}(P_i, k) < \frac{k}{2}) = 2^{-\Omega(k\sqrt{n})}$. Taking union bound over all $i \in [n]$, we have that

$$\Pr\left(\exists i \in [n] : \mathrm{RP}(P_i, k) < \frac{k}{2}\right) = 2^{-\Omega(k\sqrt{n})}.$$

This completes the proof of Lemma 6.

## E Analysis of the simultaneous simulation scheme

In this section we analyze the coding scheme in Algorithm 1 and prove Theorem 5.

We start with Section E.1, defining a notion of progress made by the simulation and analyzing it in light of error events that may happen during the simulation. Then, in Section E.2 we bound the probability that a complete simulation made insufficient progress.

### E.1 Progress analysis of Algorithm 1

We define the progress made by the simulation protocol in Algorithm 1 at a given rounds $r$ to be the number of rounds of the original protocol $\pi$ *correctly* and *simultaneously* simulated so far. In other words, we define the progress at round $r$ by

$$\mathrm{RP}(r) \triangleq \max\Big\{1 \leqslant k \leqslant r : k \text{ rounds have been correctly simultaneously simulated}\Big\}, \tag{8}$$

meaning that (a) the $\mathrm{RP}(r)$ first simulated rounds are correct, namely, $\{\mathsf{Info}(\sigma_{i,j}(r)[1,\mathrm{RP}(r)])\}_{i,j}$ is consistent with $\pi$; and (b) the simulation is simultaneous, namely, for all $i, j$, and for all $\ell \leqslant \mathrm{RP}(r)$, we have $\mathsf{Rnd}(\sigma_{i,j}(r)[\ell]) = \mathsf{Rnd}(\sigma_{1,2}(r)[\ell])$. Under this definition, the the simulation is successful whenever $\mathrm{RP}(2R) \geqslant R$.

In order to analyze the quantity $\mathrm{RP}(r)$, we first define $\mathbf{1}_{\mathrm{H}}(l), \mathbf{1}_{\mathrm{BK}}(l)$, and $\mathbf{1}_{\mathrm{E}}(l)$ to be, respectively, the indicators of whether any party sent a H, a BK, and made a tree code decoding error in round $l$. Furthermore, we shall use notations of the form $\mathbf{1}_{A\cap B}(l)$ to denote the indicator of whether $A \cap B$ occurred in round $l$. For example, $\mathbf{1}_{\mathrm{E}\cap\mathrm{BK}}(l)$ is the indicator of the event that some party made a tree code decoding error and some other party sent a BK in round $l$.

We say that after round $r$ a party $i$ is *ahead* of a party $j$ if party $i$ has simulated (possibly incorrectly) a (strictly) larger number of rounds than party $j$. Alternatively, we shall say that party $j$ is *behind* party $i$ after round $r$. Note that if $i$ is ahead of $j$ after round $r$, then in round $r+1$ we must have $\mathsf{Ind}_i(r) > \mathsf{Ind}_j(r)$. Furthermore, if party $i$ did not make any tree code decoding error in round $r+1$, then we also have $\mathsf{Ind}_i(r) > \mathsf{Ind}_{i,j}(r) = \mathsf{Ind}_j(r)$.

We now relate the progress made by the parties till the last round, i.e., $\mathrm{RP}(2R)$, with the total number of rounds with tree code decoding errors, i.e., $\sum_{l=1}^{2R} \mathbf{1}_{\mathrm{E}}(l)$ through a series of technical lemmas. We will also use $\overline{E}$ to denote the complement of the event $E$.

**Lemma 7.** *For every iteration $r \leqslant 2R$, the following holds:*

$$r \leqslant RP(r) + \sum_{l=1}^{r}\Big[\mathbf{1}_{BK}(l) + \mathbf{1}_{E}(l)\Big]. \tag{9}$$

*Proof.* We shall prove this by induction. Note that the result trivially holds for $r = 0$. We now assume that the result holds till round $r-1$. Now, note that $\sum_{l=1}^{r} \mathbf{1}_{\mathrm{BK}}(l)$ and $\sum_{l=1}^{r} \mathbf{1}_{\mathrm{E}}(l)$ are non-decreasing functions of $r$. Also, $\mathrm{RP}(r)$ can go down by at most one at each iteration. Then, there are only two ways in which the induction hypothesis can be violated at $r$. The first possibility is when all three terms in the RHS of Equation (9) remaining unchanged; the other case is when the progress goes down by one while either there was no error or there was no back. We shall complete the proof by showing that both these cases cannot happen.

Consider the first case where $\mathrm{RP}(r-1) = \mathrm{RP}(r)$ and $\mathbf{1}_{\mathrm{BK}}(r) = \mathbf{1}_{\mathrm{E}}(r) = 0$. In this case, since there were no tree code errors at iteration $r$, we have that for every $i \neq j$, $\hat{w}_{i,j}(r) = s_{j,i}(r-1)$. As a result, the parties know exactly the iterations in which an information bit was sent and iterations where a BK or H was sent. As a result, the parsed transcripts at every party now contain correct iteration numbers. Since, progress did not improve in iteration $r$, then either some party was ahead of at least one party, or there was an issue in the consistency check. In either case, at least one party will send a BK which violates the fact that $\mathbf{1}_{\mathrm{BK}}(r) = 0$. Hence, this case cannot occur.

The remaining case involves $\mathrm{RP}(r) = \mathrm{RP}(r-1) - 1$. In this case, some party must have sent a BK that caused deleting one round and decreasing the progress, hence, $\mathbf{1}_{\mathrm{BK}}(r) = 1$. Therefore, to violate the induction hypothesis, it must be that $\mathbf{1}_{\mathrm{E}}(r) = 0$. Now, as argued for the previous case, no tree code decoding error implies the parties know exactly the iterations in which an information bit was sent and iterations where a BK or H was sent. Again, in such a situation, the parties will either correctly simulate the next round if no one was ahead of the others and they were consistent, or the parties would either send a H (if they were consistent and behind the remaining parties) or a BK (if they were ahead of some party, or were inconsistent). Neither of these actions decreases progress, and hence $\mathrm{RP}(r) \geqslant \mathrm{RP}(r-1)$, which contradicts our assumption that $\mathrm{RP}(r) = \mathrm{RP}(r-1) - 1$. Therefore, this case is also vacuous, and hence Equation (9) holds by induction. □

To proceed, we define the *gap* after iteration $r$, denoted by $\mathrm{Gap}(r)$, as the maximum number of rounds incorrectly or non-simultaneously simulated by the parties. Formally,

$$\mathrm{Gap}(r) \triangleq \max_{i\in[n]} \mathsf{Ind}_i(r) - \mathrm{RP}(r).^{10}$$

Therefore, $\mathrm{Gap}(r) = 0$ implies that the parties are synchronized after iteration $r$, and every simulated round is correctly and simultaneously simulated. We shall use the convention that $\mathrm{Gap}(0) = 0$.

The next lemma relates the number of iterations where a tree code decoding error was made with the number of iterations where a BK was sent and no errors were made.

**Lemma 8.** *$\forall r \in [2R]$, it holds that*

$$\sum_{l=1}^{r} \mathbf{1}_{BK\cap\overline{E}}(l) \leqslant 2\sum_{l=1}^{r} \mathbf{1}_{E}(l). \tag{10}$$

*Proof.* First, suppose that some party makes a tree-code error in iteration $r+1$. Then the gap between the parties can increase by at most two, and thus $\mathrm{Gap}(r+1) \leqslant \mathrm{Gap}(r) + 2$. To see this, note that the maximum increase in gap occurs when a party maximizing $\mathsf{Ind}_i(r)$ decides to transmit information symbols in round $r+1$ (i.e., executes algorithm 1) while, a party minimizing $\mathsf{Ind}_i(r)$ sends a BK which deletes overall progress.

Alternately, assume no tree-code errors were made in iteration $r+1$. If $\mathrm{Gap}(r) = 0$, then all parties were consistent. Therefore, they correctly simulate the next round (execute algorithm 1) ensuring $\mathrm{Gap}(r+1) = \mathrm{Gap}(r) = 0$. On the other hand, if $\mathrm{Gap}(r) > 0$, then by the simulate-hold-rewind mechanism (algorithm 1), the parties which are ahead of at least one other party send BK and the remaining parties send either a H (if it was consistent) or a BK (due to inconsistency). In both cases, the progress remains unchanged, i.e., $\mathrm{RP}(r) = \mathrm{RP}(r+1)$, but the maximum index reduces by one, i.e., $\max_{i\in[n]} \mathsf{Ind}_i(r+1) = \max_{i\in[n]} \mathsf{Ind}_i(r) - 1$. Hence, the gap reduces exactly by one.

To summarize the above, we have that if $\mathbf{1}_{\mathrm{E}}(r) = 1$, then $\mathrm{Gap}(r+1) \leqslant \mathrm{Gap}(r) + 2$, and if $\mathbf{1}_{\mathrm{BK}\cap\overline{E}}(r) = 1$, then $\mathrm{Gap}(r+1) = \mathrm{Gap}(r) - 1$. Otherwise, $\mathrm{Gap}(r+1) = \mathrm{Gap}(r)$. Since, initially $\mathrm{Gap}(0) = 0$, we have

$$0 \leqslant \mathrm{Gap}(r+1) \leqslant 2\sum_{l=1}^{r} \mathbf{1}_{\mathrm{E}}(l) - \sum_{l=1}^{r} \mathbf{1}_{\mathrm{BK}\cap\overline{E}}(l),$$

which completes the proof. □

[10] Recall that $\mathrm{Gap}(r)$ is the gap at the end of iteration $r$, however, since it is being computed according to algorithm 1, it does not change throughout the iteration. Communication done in iteration $r$ can only affects $\mathrm{Gap}(r+1)$.

The following lemma relates progress made by the protocol to the number of rounds where at least one BK was sent.

**Lemma 9.** $\forall r \in [2R]$,

$$RP(r) \leqslant r - \sum_{l=1}^{r} \mathbf{1}_{BK}(l), \tag{11}$$

*Proof.* We prove this by induction. The base case of $r = 1$ is trivial as in the first round every party sends information bits, and hence no BK is sent. Thus, the right-hand side equals 1, while the left-hand side cannot surpass 1. Now, assume that the statement is true upto round $r-1$, i.e., we have $\mathrm{RP}(r-1) \leqslant r - 1 + \sum_{l=1}^{r-1} \mathbf{1}_{\mathrm{BK}}(l)$. Then, the statement of Lemma 9 is violated in round $r$ only if some party sent BK in round $r$ and $\mathrm{RP}(r) = \mathrm{RP}(r-1) + 1$. This situation, however, is impossible as progress cannot occur unless all parties simultaneously send an information bit; the fact that a party sent a BK immediately implies that there is no simultaneous simulation and thus no progress, in iteration $r$. The statement then follows. □

**Lemma 10.** *For every iteration $r \leqslant 2R$, we have*

$$\sum_{l=1}^{r} \mathbf{1}_{E}(l) \geqslant \frac{r - RP(r)}{5}. \tag{12}$$

*Proof.* Using Lemma 7, we have

$$\begin{aligned} 2r &\leqslant 2\mathrm{RP}(r) + 2\sum_{l=1}^{r} \mathbf{1}_{\mathrm{BK}}(l) + 2\sum_{l=1}^{r} \mathbf{1}_{\mathrm{E}}(l) \\ &\overset{(a)}{\leqslant} \mathrm{RP}(r) + r + \sum_{l=1}^{r} \mathbf{1}_{\mathrm{BK}}(l) + 2\sum_{l=1}^{r} \mathbf{1}_{\mathrm{E}}(l), \end{aligned}$$

where $(a)$ is due to Lemma 9. Therefore, we have

$$\begin{aligned} r &\leqslant \mathrm{RP}(r) + \sum_{l=1}^{r} \mathbf{1}_{\mathrm{BK}}(l) + 2\sum_{l=1}^{r} \mathbf{1}_{\mathrm{E}}(l) \\ &\leqslant \mathrm{RP}(r) + \sum_{l=1}^{r} \mathbf{1}_{\mathrm{BK}\cap\mathrm{E}}(l) + \sum_{l=1}^{r} \mathbf{1}_{\mathrm{BK}\cap\overline{E}}(l) + 2\sum_{l=1}^{r} \mathbf{1}_{\mathrm{E}}(l) \\ &\leqslant \mathrm{RP}(r) + 5\sum_{l=1}^{r} \mathbf{1}_{\mathrm{E}}(l), \end{aligned}$$

where the last inequality follows using Lemma 8 and noting that $\sum_{l=1}^{r} \mathbf{1}_{\mathrm{BK}\cap\mathrm{E}} \leqslant \sum_{l=1}^{r} \mathbf{1}_{\mathrm{E}}(l)$. □

Lemma 10 relates the number of iterations with tree code decoding errors till iteration $r$ and the progress made so far. To complete the proof of Theorem 5, it remains to suitably bound the probability of error in Algorithm 1, by bounding the number of iterations with tree code decoding errors. This is dealt with in the following section.

### E.2 Success Probability of Algorithm 1

In this section, we bound the probability of error of the simultaneous simulation scheme in Algorithm 1. Recall that the simulation fails if either it returns incorrect transcripts of the original protocol $\pi$, or if certain rounds of $\pi$ were not simultaneously simulated. Let $\mathcal{E}_1$ denote the event that $\mathrm{RP}(2R) < R$, i.e., that the simulation did not correctly and simultaneously simulated all $\pi$'s rounds during the first $2R$ iterations. Further, let $\mathcal{E}_2$ denote the event that some party made a tree code decoding error of length larger than $R$ after the completion of $3R$ rounds. We now bound the probability of these two event, since if both do not happen, the simulation succeeds.

*Overview.* We begin our analysis by providing a bound on the number of A2A errors required in the simulation protocol of Algorithm 1 to ensure that $\mathcal{E}_1$ occurs. To do so, we first claim that for $\mathcal{E}_1$ to occur, out of the set of $2R$ iterations, one can find disjoint sets of consecutive iterations, referred to as bad runs (precisely defined later), where one party makes tree code decoding errors in all of those iterations. The total number of iterations taking part in these bad runs is shown to be at least $\frac{R}{5}$ given $\mathcal{E}_1$ occurs. By Lemma 3, in each bad run, the party making the tree code decoding errors must make A2A decoding errors in at least one-fourth of the iterations involved in that particular bad run. We bound this probability which then gives us a bound on $\Pr(\mathcal{E}_1)$. To complete the proof of Theorem 5, we then bound $\Pr(\mathcal{E}_2)$ using Lemma 3 and the probability of A2A errors.

*Notations.* Before we proceed, we set up some notation. Assume that party $P_i$ makes a tree code error in iteration $r$ Recall that in any round, $P_i$ decodes one tree-coded string from each of its neighbours. Hence, a tree-code error in iteration $r$ can stem from incorrectly decoding one or more of these $n-1$ tree-coded strings. Now, choose the string for which $P_i$ made the tree code decoding error of maximum length, and denote this length by $t$. Then we say that $P_i$ has a *bad suffix* of length $t$ at iteration $r$.

Next, we define a *run* $I \subseteq [2R]$ to be a set of *consecutive* iterations. The run is said to start at iteration $\min I$ and end at iteration $\max I$; the *length* of the run, $I = 1 + \max I - \min I$ is the cardinality of the set $I$. Finally, a run $I$ is said to be a *bad-run* if there exists some $j \in [n]$ such that $P_j$ has a bad suffix of length $|I|$ at iteration $\max I$ where the run ends.

*A bound on* $\Pr(\mathcal{E}_1)$. We start with a technical lemma showing that if $\mathcal{E}_1$ occurs, we can lower bound the total length of some disjoint bad-runs.

**Lemma 11.** *If $\mathcal{E}_1$ occurs, then there exists a disjoint set of $K \geqslant 1$ bad-runs $I_1, I_2, \ldots, I_K$ satisfying $\sum_{k=1}^{K} |I_k| \geqslant \frac{R}{5}$. Furthermore, if iteration $l \notin \bigcup_{k=1}^{K} I_k$ then $\mathbf{1}_E(l) = 0$, i.e., no party makes a tree code decoding error in iteration $l$.*

*Proof.* We construct the bad runs $I_1, \ldots, I_K$ starting from the last iteration as follows. Let $r \leqslant 2R$ be the last iteration where some party made a tree code decoding error. Among the parties that made a tree code decoding error at iteration $r$, let $P_i$ be the party with the longest bad suffix at iteration $r$. Let the length of this bad suffix be $t$. Then, define $I_1 \triangleq \{r-t+1, \ldots, r\}$, and observe that $I_1$ is indeed a bad run.

To construct the remaining bad runs $I_2, \ldots, I_K$ follow the same procedure starting from iteration $r-t$. This process terminates when either all iterations have been used up, or we reach some iteration $r'$ such that $\mathbf{1}_{\mathrm{E}}(l) = 0$ for every $l \leqslant r'$.

Note that the bad runs $I_1, \ldots, I_K$ constructed in this manner are indeed disjoint, and for every iteration $l \notin \cup_{k=1}^{K} I_k$, we have $\mathbf{1}_{\mathrm{E}}(l) = 0$. It remains to show that $\sum_{l=1}^{k} |I_l| \geqslant \frac{R}{5}$, conditioned on the occurrence of $\mathcal{E}_1$.

To see this, since $\{r \in [2R] \mid \mathbf{1}_{\mathrm{E}}(r) = 1\} \subseteq \bigcup_{l=1}^{k} I_l$, Lemma 10 then gives

$$\sum_{l=1}^{k} |I_l| \geqslant \sum_{r=1}^{2R} \mathbf{1}_{\mathrm{E}}(r) \geqslant \frac{2R - \mathrm{RP}(R)}{5}.$$

If $\mathcal{E}_1$ occurs we have $\mathrm{RP}(2R) < R$, yielding,

$$\sum_{l=1}^{k} |I_l| \geqslant \frac{R}{5}.$$

□

With the help of Lemma 11 we can show that if the simulation did not succeed, this must be due to many iteration in which the A2A scheme failed to provide a resilient exchange of information.

**Lemma 12.** *Given that $\mathcal{E}_1$ has occurred, there exists a set $I \subseteq [2R]$ of length $|I| \geqslant \frac{R}{20}$, such that there is an* A2A *error in every iteration $r \in I$.*

*Proof.* Consider the bad-runs $I_1, \ldots, I_K$ given by Lemma 11. For any $k \in [K]$ let $M_k = \max I_k$. Then, by construction, some party $P_i$ has a bad suffix of length $|I_k|$ at iteration $M_k$. Since the tree code has distance $\alpha = \frac{1}{2}$, Lemma 3 says that there must be a subset $J_k \subseteq I_k$ satisfying $|J_k| \geqslant \frac{1}{4}|I_k|$, such that $P_i$ made an A2A decoding error for every round in $J_k$. The result then follows by defining $I = \cup_{k=1}^{K} J_k$, and noting that Lemma 11 implies that $|I| \geqslant \frac{1}{4}\sum_{k=1}^{K} |I_k| \geqslant \frac{R}{20}$. □

Since a decoding error in A2A has a small probability of $2^{-\Omega(\sqrt{n})}$ (Theorem 3), the existence of many iterations with decoding errors in A2A also has a small probability, which in turn bounds the probability of the occurrence of the event $\mathcal{E}_1$.

**Lemma 13.**

$$\Pr(\mathcal{E}_1) \leqslant 2^{-\Omega(R\sqrt{n})}. \tag{13}$$

*Proof.* Due to Lemma 12, to bound $\Pr(\mathcal{E}_1)$, we can bound the probability to have $R/20$ iterations with A2A decoding error. Recall from Theorem 3 that the probability of an A2A decoding error in any iteration is $2^{-\Omega(\sqrt{n})}$, and since the noise is independent, the failure probabilities of different A2A instances (i.e., different iterations) are independent. Then, probability that there exists a set of iterations $I \subseteq [2R]$ of length $|I| \geqslant \frac{R}{20}$ such that some party makes an A2A decoding error in iteration $l \in I$, is upper bounded by

$$\sum_{k=\frac{R}{20}}^{2R} \binom{2R}{k} 2^{-k\Omega(\sqrt{n})} \leqslant \sum_{k=\frac{R}{20}}^{2R} \binom{2R}{k} 2^{-R\Omega(\sqrt{n})} \leqslant 2^{2R} 2^{-R\Omega(\sqrt{n})} = 2^{-R\Omega(\sqrt{n})}. \tag{14}$$

Then, by Lemma 12, we have $\Pr(\mathcal{E}_1) \leqslant 2^{-R\Omega(\sqrt{n})}$. □

*A bound on* $\Pr(\mathcal{E}_2)$. To complete the error probability analysis, we now bound the probability of the event $\mathcal{E}_2$.

**Lemma 14.**

$$\Pr(\mathcal{E}_2) \leqslant 2^{-\Omega(R\sqrt{n})}. \tag{15}$$

*Proof.* To incorrectly decode the first $2R$ symbols of the $3R$-long tree-coded string, a party must have a bad suffix of length at least $R+1$ after $3R$ iteartions.

Recall that the tree code $\mathcal{T}$ has a distance $\alpha = \frac{1}{2}$. Thus, by Lemma 3, a party having a bad suffix of length $\ell$ must have made at least $\frac{\ell}{4}$ A2A decoding errors in the last $\ell$ iterations. Moreover, since the A2A decoding error event includes any one of the $n$ parties making an error, the event that there exists a party with a bad suffix of length $\ell$ implies that at least $\frac{\ell}{4}$ of the last $\ell$ iterations had A2A decoding errors.

Theorem 3 shows that the probability of an A2A decoding error is $2^{-\Omega(\sqrt{n})}$. Then, the probability that there exists a party with a bad suffix of length $R+1$ at the end of Algorithm 1 can be upper bounded by

$$\sum_{\ell=R+1}^{3R} \binom{\ell}{\frac{\ell}{4}} 2^{-\ell\Omega(\sqrt{n})} \leqslant \sum_{\ell=R+1}^{3R} 2^{\ell} 2^{-R\Omega(\sqrt{n})} \leqslant 2R2^{3R}2^{-R\Omega(\sqrt{n})} \leqslant 2^{-R\Omega(\sqrt{n})}.$$

Therefore,

$$\Pr(\mathcal{E}_2) \leqslant 2^{-R\Omega(\sqrt{n})}. \qquad \square$$

*Completing the proof of Theorem 5.* We now combine all the technical results so far to complete the proof of Theorem 5.

*Proof of Theorem 5.* Since Algorithm 1 takes $3R = 3|\pi|$ iterations, where each iteration executes exactly a single A2A instance, which in turn takes $O(1)$ rounds of communication, and no further communication occurs, the simulation takes $3R \times O(1) = O(|\pi|)$ rounds of communication.

Following Lemma 13 and Lemma 14, the failure probability of the simulation can be upper bounded by

$$\Pr(\text{error in Alg 1}) \leqslant \Pr(\mathcal{E}_1) + \Pr(\mathcal{E}_2) \leqslant 2 \cdot 2^{-\Omega(R\sqrt{n})} = 2^{-\Omega(R\sqrt{n})},$$

This completes the proof. $\square$

## F Merged IC: A short overview

The IC scheme stated in Theorem 6 merges the chunked nature of ChunkRS with the capability of simultaneous simulation. A chunk size $c = O(1)$ is considered, and the protocol simulation proceeds in chunks of $c$ rounds. The protocol alphabet is thus chosen as $S = \{\{0,1\}^c, \mathrm{BK}, \mathrm{H}\}$, and we shall use a tree code $\mathcal{T} : S^* \to \Sigma^*$ with distance $\alpha = \frac{1}{2}$ to protect the transmissions.

As in the two previous schemes, all parties maintain their sent and received transcripts $(s_{i,j}, w_{i,j})_{j \neq i}$. Then, at the beginning of any chunk, party $i$ first decodes all the symbols received from party $j$ in the previous chunks to obtain the string $\hat{w}_{i,j}$. Then, the parties parse these strings according to the

parsing scheme used in the simultaneous simulation scheme, but in a chunk-by-chunk instead of a round-by-round fashion. This is then followed by a consistency check to determine whether a party believes that the simulation so far has been consistent and simultaneous.

Following this, the parties begin the simulation of the next chunk as in ChunkRS. As in ChunkRS, if the parties failed the consistency check, they send dummy 0s throughout the simulation of the next chunk and send a BK to the remaining parties after the simulation. However, the key difference with the simulation of the next chunk in ChunkRS and the merged IC lies in the situation where the party believes that no party is behind it, but some parties are strictly ahead of it. In such a situation, in ChunkRS, the party would have continued with the simulation and then sent the simulated chunk to the remaining parties. However, in the merged IC, even in this scenario, the party will start sending dummy 0s during the chunk simulation and then send a H to the remaining parties at the end of the chunk simulation.

If the original (noiseless) protocol had $R$ rounds, then this process continues for $\frac{2R}{c}$ chunked rounds. Following this, to avoid the decoding error at the end which can hamper simultaneous simulation, every party sends additional $R$ zeros to the remaining parties. Then, every party makes a fresh tree code decoding of the symbols received thus far, truncates them to a size $\frac{2R}{c}$, and then returns the relevant parsed strings truncated to $R$ bits.

The proof technique is similar to that of the scheme with simultaneous simulation. We analyze the progress made by the parties and relate it to the number of chunked rounds with tree code decoding errors. Next, we show that for the global progress to be lower than $\frac{R}{c}$ even after the $\frac{2R}{c}$ chunked round, there exists a large number of chunked rounds where at least one party made a tree code decoding error. Using this, we shall be able to show that there exists a history where a large number of $\mathsf{A2A}$ decoding errors were made, and this would be a low-probability event. As a result, we shall be able to show that the global progress must have been large enough with high probability. The additional dummy rounds ensure that all parties are able to successfully return the correct transcripts unless there is a tree code error of large length, which again is a low probability event. To avoid repetitive proofs, we omit the details.

# G Missing MPC Definition/Theorems/Proofs

## G.1 MPC Definition over BSC

**Definition 5 (Semi-Honest Security over $\mathsf{BSC}_\varepsilon$ Channels).** *An $n$-party protocol $\pi$ for computing a function $f$ is said to be statistically secure against semi-honest adversaries over $\mathsf{BSC}_\varepsilon$ if, for any $\mathcal{A} \subset [n]$, for every real-world adversary corrupting a set $\mathcal{A}$ of parties, there exists an efficient simulator $\mathsf{Sim}^{\mathsf{BSC}}$ such that, for all inputs $x$ and statistical security parameter $\kappa$, the following holds:*

$$\left\{\mathsf{Sim}^{\mathsf{BSC}}(\mathcal{A}; 1^\kappa, x_\mathcal{A}, f_\mathcal{A}(x)), f(x)\right\} \approx_{\exp(-\kappa)} \left\{\mathsf{View}_\pi^{\mathsf{BSC}}(\mathcal{A}; 1^\kappa, x), \mathsf{Output}_\pi^{\mathsf{BSC}}(1^\kappa, x)\right\}.$$

We can also define the stronger security notion called *semi-honest security with channel transparency over* $\mathsf{BSC}$, where the simulator can generate the augmented view of the corrupted parties, such that the joint view of the augmented view over $\mathsf{BSC}$ (represented by $\mathsf{View}_\pi^{\mathsf{BSC},*}$) and the protocol output are indistinguishable from the simulator output and the ideal output.

## G.2 NAND security Theorem [GMW87, Gol04]

**Theorem 10.** *The $n$-party $\mathsf{NAND}$ protocol $\Pi = \Pi_{\mathsf{NAND}}$ is channel transparent and it securely implements the $\mathsf{NAND}$ functionality over non-noisy channels, i.e. there exists a simulator $\mathsf{Sim}$, such*

*that for any set of corrupt parties $\mathcal{A} \subset [n]$ and any inputs $\{x_i, y_i \in \mathbb{Z}_2\}_{i\in[n]}$ the following holds:*

$$(\mathsf{Sim}(\mathcal{A}; \{x_i, y_i, z_i\}_{i\in\mathcal{A}}), \{z_i\}_{i\in[n]}) \equiv (\mathsf{View}^*_{\Pi}(\{x_i, y_i\}_{i\in[n]}), \mathsf{Output}_{\Pi}(\{x_i, y_i\}_{i\in[n]}))$$

*where*

$$\text{For } i \in [n] : z_i \leftarrow_\$ \{0,1\} \text{ conditioned on } \quad \sum_{i=1}^{n} z_i = 1 + \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right).$$

### G.3 Proof of Theorem 8

*Proof.* The simulator description is presented in Figure 3. The Simulator essentially simulates the A2A NAND protocol $\mathsf{A2A}(\Pi_{\mathsf{NAND}})$ for each corrupt party, where each message sent by an honest party is a freshly sampled random bit.

The augmented view of the $C$ corrupt parties in the real protocol execution is defined as

$$\mathsf{View}^{\mathsf{BSC},*}_{\Pi'}(\mathcal{A}; x, y) = \left\{x_{\mathcal{A}}, r_{\mathcal{A}}, \mathsf{corr}_{\mathcal{A}}, M_1, M_2, \ldots, M_n\right\}$$

where $M_i$ are the messages received by party $i$ during the execution of the protocol, and corr correspond to 2 instances of pairwise random OT correlations.

We show the simulator is perfectly indistinguishable from the view of the A2A NAND protocol by showing that the input to the A2A protocol has the same distribution in both the protocol and the simulator.

For any $i, j \in [n]$ with $i < j$ parties $P_i$ and $P_j$ execute two instances of OT in the NAND protocol, where the first instance has the following form: $P_i$ acts as the receiver with bit $x_i$ and it has $(u_{b'}, b')$ OT correlation. Party $P_j$ acts as the sender with messages $(m_0, m_1) = (r_1^{ij}, y_i + r_1^{ij})$ with OT correlation $(u_0, u_1)$,

- If $i, j \in \mathcal{A}$: the simulator $\mathsf{Sim}^{\mathsf{BSC}}_{\mathsf{NAND}}$ picks the same messages as input to A2A as the protocol.
- If $i \in \mathcal{A}, j \notin \mathcal{A}$: $P_i$'s input messages to A2A can be exactly simulated. In the protocol, $P_j$ sends $(m_0 + u_{c'}, m_1 + u_{1-c'})$ to $P_i$ via A2A. This can be simulated by a pair of random independent bits, where the randomness is over $u_{1-b}$ and $r_i^{ij}$ - random bits not in the view of $P_j$.
- If $i \notin \mathcal{A}, j \in \mathcal{A}$: $P_j$'s input messages to A2A can be exactly simulated. $P_i$'s only input message to A2A in the protocol for $P_j$ is the bit $(b + b')$. This can be simulated by a random bit, since $b'$ is independent of all other messages sent in the protocol.
- $i, j \notin \mathcal{A}$: With the same argument as the above two cases, the input messages of parties $P_i$ and $P_j$ can be simulated by random bits.

The same argument also shows that the view of the second OT execution via A2A has the same distribution in both the protocol and the simulator.

□

### G.4 Security Proof for $\Pi^{\mathsf{BSC}}_{\mathsf{MPC}}$

We now prove that protocol $\Pi^{\mathsf{BSC}}_{\mathsf{MPC}}$ (in Figure 6) achieves statistical security against semi-honest adversaries over $\mathsf{BSC}_\varepsilon$ channels. The proof proceeds by constructing a simulator $\mathsf{Sim}^{\mathsf{BSC}}_{\mathsf{MPC}}$ that, given the corrupt parties' inputs and the ideal output, produces a view indistinguishable from the real protocol execution.

**Protocol $\Pi_{\mathsf{MPC}}$**

- **Inputs:** A (public) circuit $f = C$ and inputs $x_1, x_2, \ldots, x_n$ held by party $1, 2, \ldots, n$, respectively.
- **Initialization:**
  - For every party $1 \leqslant i \leqslant n$, $i$ sends its encoded share of its inputs $x_{ij}$ using an $\mathsf{ECC}$ (guaranteed by Lemma 1) of length $O(\kappa) + \log(n^2 m)$, for $1 \leqslant j \leqslant m$. It also receives and decodes the (encoded) shares of other parties.
- **Evaluation:**
  - Let $\pi_f$ be the GMW protocol that describes the evaluation of the NAND gates, where the parties evaluate each of the NAND gates in a predetermined topological order.
  - Apply the interactive coding scheme of Theorem 6 with chunk size $c = 6$ for the evaluation of $\pi_f$ with correlated randomness $\mathsf{ROT}$. The chunk size of 6 ensures that exactly one NAND gate is simulated in each chunk of the protocol execution.
- **Reconstruction:**
  - If the interactive coding does not succeed, all parties output $\perp$.
  - Otherwise, for each output gate $g$, all parties send their shares to each other using $\mathsf{ECC}$ of length $O(\kappa)$.
  - Each party obtains the expected output by adding its share to the received shares for each output gate.

Fig. 6: Protocol $\Pi_{\mathsf{MPC}}$

*Helper functions.* To formalize the simulator, we define three deterministic functions that capture each party's local computation during the IC scheme:

- $g_i \leftarrow \mathsf{nextNANDgate}(t, \mathsf{rcvMsg}_i)$: Given iteration index $t$ and party $P_i$'s received messages, outputs the index of the NAND gate that $P_i$ will execute in iteration $t$. This function depends only on the transcript structure dictated by the IC scheme, not on $P_i$'s private input.
- $(a_i, b_i) \leftarrow \mathsf{nextNANDinp}(t, x_i, r_i, \mathsf{corr}_i, \mathsf{rcvMsg}_i)$: Outputs party $P_i$'s input shares to the NAND gate in iteration $t$. These shares depend on $P_i$'s original input $x_i$, its internal randomness $r_i$, the correlated randomness $\mathsf{corr}_i$, and all previously received messages $\mathsf{rcvMsg}_i$.
- $s_i \leftarrow \mathsf{nextSymbol}(t, \mathsf{rcvMsg}_i)$: Outputs the tree code symbol that $P_i$ sends in iteration $t$. This is a deterministic function of the received messages that encodes $P_i$'s view of the protocol's progress.

Crucially, only $\mathsf{nextNANDinp}$ depends on private inputs. The functions $\mathsf{nextNANDgate}$ and $\mathsf{nextSymbol}$ depend only on the received messages. We also define a helper function $\mathsf{SameOutGate}$, which takes as input a set of $n$ gates and a circuit $C$ and outputs a boolean value $\mathsf{true}$ if all input gates match and they are an output gate in $C$, else the function returns $\mathsf{false}$.

Figure 7 presents the formal simulator for protocol $\Pi^{\mathsf{BSC}}_{\mathsf{MPC}}$. At a high level, the simulator proceeds as follows:

- **Initialization phase**: Any subset of less than or equal to $n-1$ shares of an $(n, n)$-additive secret sharing is identically distributed for any two secrets. Hence, the simulator essentially follows the same protocol as the initialization phase of $\Pi^{\mathsf{BSC}}_{\mathsf{MPC}}$, but with honest party inputs replaced by random bits.

– **Evaluation phase**: 'The simulator executes $|\pi_f|/3$ iterations/chunks, where in each iteration it simulates a single A2A-NAND execution, where $\pi_f$ is the GMW evaluation protocol. Let $T$ represent the total number of bits of communication in the protocol $\mathsf{A2A}(\Pi_{\mathsf{NAND}})$. Then, the simulator first samples $T$ bits of noise (each an element of $\mathsf{bernoulli}(\varepsilon)$). When this sampled noise is not all zeroes and not all parties are simulating the same NAND gate (as decided by nextNANDgate), the simulated view is generated using $\mathsf{Sim}^{\mathsf{BSC}}_{\mathsf{NAND}}$ given corrupt party inputs. If all parties execute the same output gate (say $g$) and no BSC noise is added, then we simulate the protocol views using $\mathsf{Sim}_{\mathsf{NAND}}$ given corrupt party inputs and ideal output of gate $g$. The tree code symbols exchanged via A2A are a deterministic function of the received messages alone. Hence, the simulator can also generate this view, provided the received messages are simulated for all parties.
– **Reconstruction phase**: For each output gate, parties send ECC encoding of their output gate shares as the coding scheme dictates.

**Simulator** $\mathsf{Sim}^{\mathsf{BSC}}_{\mathsf{MPC}}(\mathcal{A}; x_{\mathcal{A}}; y = C(x))$

**Initialization**

1. **For** each $i \in [n]$:
   – Initialize $\mathsf{rcvMsg}_i = \emptyset$
   – Sample random bits $x_{i,j} \leftarrow_{\$} \{0,1\}$ for $j \in [n-1]$
   – **if** $i \in \mathcal{A}$: Set $x_{i,n-1}$ such that $x_i = \sum_{j \in [n]} x_{i,j}$
   – **Else**: Set $x_{i,n-1} \leftarrow_{\$} \{0,1\}$
2. For all $i, j \in [n]$, $i \neq j$: append to $\mathsf{rcvMsg}_i$: $\mathsf{ECC}(x_{ij}) + \mathcal{N}$ as the received messages from $P_j$, where ECC is an error-correcting code with length $O(\kappa) + \log(mn^2)$ (see Lemma 1) and $\mathcal{N} \leftarrow \mathsf{bernoulli}(\varepsilon)$.

**Evaluation**

1. Simulate a single chunk of $c = 6$ rounds i.e a single $\Pi_{\mathsf{NAND}}$ execution: For each $P_i$, do the following for $l = 1$ to $\frac{2|\pi_f|}{c}$:
   (a) For NAND execution in the current chunk:
   (b) For $i \in [n]$: $g_i \leftarrow \mathsf{nextNANDgate}(\mathsf{rcvMsg}_j)$
      i. For each corrupt part $j \in \mathcal{A}$: $(a_j, b_j) \leftarrow \mathsf{nextNANDinp}(x_j, r_j, \mathsf{rcvMsg}_j)$
      ii. Initialize $T$-bit string $r$: for each $i$: $r_i \leftarrow \mathcal{N}(\varepsilon)$
      iii. If $\neg\mathsf{SameOutGate}(C, \{g_i\}_{i \in [n]}) \vee \ (r \neq 0^T)$ :
         – Update $\mathsf{rcvMsg}_i$ with the output of $\mathsf{Sim}^{\mathsf{BSC}}_{\mathsf{NAND}}(\mathcal{A}; \{a_i, b_i\}_{i \in \mathcal{A}})$
      iv. Else:
         – Let $y_g$ be the ideal output corresponding to gate $g$
         – For $i \in [n]$, sample $o^g_i \leftarrow_{\$} \{0,1\}$ conditioned on $\sum_{i=1}^{n} o^g_i = y_g$
         – Update $\mathsf{View}_i$ with the output of $\mathsf{Sim}_{\mathsf{NAND}}(\mathcal{A}; \{a_i, b_i\}_{i \in \mathcal{A}}; \{o^g_i\}_{i \in \mathcal{A}})$
      v. $t_i \leftarrow \mathsf{nextSymbol}(\mathsf{rcvMsg}_i)$
      vi. Simulate $\log|\Gamma|$ calls to A2A protocol where party $P_i$ sends to party $P_j$ bits in the tree code symbol $t_{ij}$ for each $i \neq j$
      vii. For each $i$, update $\mathsf{rcvMsg}_i$ with the views for $P_i$ in the above $\log|\Gamma|$ calls to A2A protocol

**Output Calculation**

1. If too many errors occur and the simulated transcript's length is less than $|\pi|$ rounds, output $\perp$.
2. For each output gate $g$: Each $P_i$ sends $\mathsf{ECC}(o_i^g) + \mathcal{N}$ where the noise $\mathcal{N} \leftarrow \mathsf{bernoulli}(\varepsilon)$.

Fig. 7: Simulator $\mathsf{Sim}^{\mathsf{BSC}}_{\mathsf{MPC}}$

Let $\widetilde{\mathsf{View}}^{\mathsf{BSC}}_{\mathsf{MPC}}(\mathcal{A}; \kappa, x, t)$ represent the view of the corrupted parties $\mathcal{A}$ in the protocol $\Pi_{\mathsf{MPC}}$ with inputs $x$ after executing exactly $t$ iterations/chunks of the evaluation phase. Likewise, let $\widetilde{\mathsf{Sim}}^{\mathsf{BSC}}_{\mathsf{MPC}}(\mathcal{A}; \kappa, x_{\mathcal{A}}, f(x), t)$ represent the view generated by the simulator $\mathsf{Sim}^{\mathsf{BSC}}_{\mathsf{MPC}}$ until $t$ iterations of the evaluation phase. We next prove the following helper lemma and then the main security theorem.

**Lemma 15.** *For every $x \in \{\{0,1\}^m\}^n$, security parameter $\kappa, \mathcal{A} \subset [n]$ and $t \in [|\pi|/3]$:*

$$\widetilde{\mathsf{View}}^{\mathsf{BSC}}_{\mathsf{MPC}}(\mathcal{A}; x, t) \approx_{\exp(-\kappa)} \widetilde{\mathsf{Sim}}^{\mathsf{BSC}}_{\mathsf{MPC}}(\mathcal{A}; x_{\mathcal{A}}, f(x), t)$$

*Proof.* We prove this by inducting over $t$.

- Base case $t = 0$ corresponds to the simulator and the protocol views being statistically indistinguishable after the initialization phase. In this phase, the view of all corrupt parties in a real protocol execution consists of $|\mathcal{A}|$ shares of each party's input, encoded with an $\mathsf{ECC}$ of size $O(\kappa)$. Since we use the standard $(n, n)$-additive secret sharing, $\leqslant n-1$ shares of any two secrets are perfectly indistinguishable. Hence, the protocol view is identical to that of the simulated view before the start of the evaluation phase.
- Let the statement hold for any $t = k-1$ for $0 < k < |\pi_f|/3$. We further show it holds for $t = k$.
- $\widetilde{\mathsf{View}}^{\mathsf{BSC}}_{\Pi}(\mathcal{A}, x, k)$ includes $\widetilde{\mathsf{View}}^{\mathsf{BSC}}_{\Pi}(\mathcal{A}, x, k-1)$ and the messages sent during execution of $k^{th}$ chunk. In this $k^{th}$ chunk of the protocol, the parties execute an A2A-NAND gate (say $g$, with real or dummy inputs), then exchange tree code symbols via $\mathsf{A2A}$. The simulator generates the following view for the NAND execution in the $k^{th}$ chunk of the protocol:
  - For each party $P_i$ it uses the variable $\mathsf{rcvMsg}_i$ to compute next gate $g_i$ to be evaluated (using $\mathsf{nextNANDgate}$) and the corresponding inputs $(a_i, b_i)$ (using $\mathsf{nextNANDinp}$ only for honest parties). By induction, $\mathsf{rcvMsg}_i$ is indistinguishable from the received messages by party $P_i$ in the protocol. This implies the NAND inputs generated by $\mathsf{nextNANDinp}$ are indistinguishable for both the protocol and the simulator.
  - By construction, each NAND execution uses independent $\mathsf{ROT}$ correlations. Hence, the A2A-NAND protocol view in the $k^{th}$ chunk is perfectly indistinguishable from (i) the output of $\mathsf{Sim}^{\mathsf{BSC}}_{\mathsf{NAND}}$ from Theorem 8 with inputs $\{a_i, b_i\}_{i \in \mathcal{A}}$, and also from (ii) the output of $\mathsf{Sim}_{\mathsf{NAND}}$ from Theorem 7 with the same inputs and any arbitrary NAND output shares.
- Both the protocol and the simulator have statistically indistinguishable distributions for $\mathsf{rcvMsg}_i$ until the NAND execution in the $k^{th}$ iteration. The function $\mathsf{nextSymbol}$ is a deterministic function of each party's $\mathsf{rcvMsg}_i$. Hence, the simulator can generate the tree symbols input to the $\mathsf{A2A}$ protocol as described in Step (v) - indistinguishable from the $\mathsf{A2A}$ tree symbols sent in the protocol.

□

**Theorem 11 (Main theorem, Formal).** *Fix a security parameter $\kappa > 0$ and a noise parameter $\varepsilon \in (0, \frac{1}{2})$. For any circuit $C : \{\{0,1\}^m\}^n \to \{0,1\}^{m'}$ (that's poly-sized in $\kappa$), there exists a $n$-party*

*protocol $\Pi_{\mathsf{MPC}}$ (presented in Protocol 6) that evaluates the circuit $C(x_1, \ldots, x_n)$ in the $\mathcal{F}_{\mathsf{ROT}}$ hybrid model over $\mathsf{BSC}_\varepsilon$ channels that is statistically secure against semi-honest adversaries. Moreover, the communication complexity of $\Pi_{\mathsf{MPC}}$ is $O_\varepsilon(n^2(|C| + \kappa))$ for $m, m' = O(|C|)$ and $n = O(\mathrm{poly}(\kappa))$.*

*Proof.* Since we consider deterministic functionality, we can analyze the correctness and privacy separately.

- **Correctness**: By the correctness of the IC scheme (Theorem 6), the evaluation phase of protocol $\Pi^{\mathsf{BSC}}_{\mathsf{MPC}}$ with ROT correlated randomness exactly matches the output of the GMW protocol's evaluation phase for some correlated randomness. Protocol $\Pi^{\mathsf{BSC}}_{\mathsf{MPC}}$ also simulates the original GMW protocol in the initialization and reconstruction phases, where each message is encoded using $\mathsf{ECC}$, which is incorrectly decoded with probability $2^{-O(\kappa)}$. Hence, the correctness of $\Pi^{\mathsf{BSC}}_{\mathsf{MPC}}$ reduces to the correctness of the original GMW protocol.
- **Privacy**: By Lemma 15, the views of the corrupted parties in the protocol and the simulator are indistinguishable until the end of the evaluation phase with $1 - 2^{-O(\kappa)}$ probability. In the reconstruction phase, the view of all parties in a real execution of the protocol consists of all the $n$ shares of each output gate (say $g$), encoded with an $\mathsf{ECC}$ of size $O(|C| + \kappa)$. The IC scheme ensures that the final execution of the output NAND gate $g$ was carried out in the same iteration (say $t_g$) by all parties. By the correctness of the IC scheme, no noise is added in iteration $t_g$. In $\mathsf{Sim}^{\mathsf{BSC}}_{\mathsf{MPC}}$, we use the simulator $\mathsf{Sim}_{\mathsf{NAND}}$ to generate the $\mathsf{A2A\text{-}NAND}$ views for precisely the same iterations - when all parties execute the same output gate, and no $\mathsf{BSC}$ noise is added. Hence, the $n$ shares for each output gate in the simulator and the protocol correspond to the NAND gate's output from the same iteration, making the joint view of the reconstruction and evaluation phases indistinguishable in both the simulator and the protocol.
- **Communication complexity**: The initialization phase has communication cost $O_\varepsilon\big(n^2 m(\kappa + \log(mn^2))\big) = O_\varepsilon\big(n^2 m(\kappa + \log m + \log n)\big)$. Similarly the reconstruction phase has communication cost $O_\varepsilon\big(n^2 m'(\kappa + \log m' + \log n)\big)$. Since the underlying IC scheme is constant-rate, the evaluation phase has communication cost $O_\varepsilon\big(n^2(|C| + \kappa)\big)$. Combining all three phases, the total communication complexity for $m, m' = O(|C|)$ and $n = O(\mathrm{poly}(\kappa))$ simplifies to:

$$\mathsf{CC}(\Pi^{\mathsf{BSC}}_{\mathsf{MPC}}) = O_\varepsilon\big(n^2(|C| + \kappa)\big)$$

□

## H Extensions and Further Optimizations

We now discuss several extensions to our results and compare them with other approaches.

*MPC with private outputs.* We can naturally extend the MPC construction in Theorem 11 with public outputs so that only specific parties get specific output bits. Let $C(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ be a circuit with inputs $x_i \in \{0,1\}^m$ and outputs $y_i \in \{0,1\}^{m'}$. If we want to design an MPC protocol over $\mathsf{BSC}$ channels such that only $P_i$ learns $y_i$, we can use the construction in Theorem 11 where each party $P_i$ inputs $(x_i, r_i)$, where $r_i$ is a random string of length $m'$. And we use the MPC protocol to compute the circuit $C^*((x_1, r_1), \ldots, (x, r_n)) = ((y_1 + r_1), \ldots, (y_n + r_n))$ where $C(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$.

Since only party $P_i$ knows the random string $r_i$, only $P_i$ can learn the output $y_i$ given the public output of $C^*$ from the MPC protocol. It's also easy to verify that this transformation doesn't affect the MPC protocol's rate.

*Reducing the round complexity.* For any boolean circuit $C$, the classic GMW protocol gives a secure MPC protocol over noiseless channels with round complexity $O(\mathsf{depth}(C))$. Whereas our GMW-inspired MPC protocol over $\mathsf{BSC}$ channels has round complexity $O(|C|)$, since it sequentially simulates one NAND gate per iteration of the IC protocol. We now describe a transformation that reduces the round complexity to $O(\kappa \cdot (\mathsf{depth}(C) + \kappa))$ while preserving the same asymptotic communication complexity as GMW:

We decompose $C$ into layers, where layer $d$ contains all gates at depth $d$. We classify each layer as either **shallow** for having fewer than $\kappa$ gates, or **wide** for having at least $\kappa$ gates.

We then partition the layers into maximal contiguous groups of the same type. Formally, let $C_1, C_2, \ldots, C_\ell$ be the sequence of sub-circuits obtained by the following greedy procedure:

1. Start at layer 1. Initialize $i = 1$.
2. Let $C_i$ contain all consecutive layers starting from the current position that have the same type (all shallow or all wide).
3. Advance to the next layer after $C_i$ and increment $i$.
4. Repeat until all layers are assigned.

This procedure is deterministic and produces a unique decomposition. Note that:

- If all layers are wide, the decomposition consists of a single wide sub-circuit $C_1$.
- If all layers are shallow, the decomposition consists of a single shallow sub-circuit $C_1$.
- In general, the sub-circuits alternate between shallow and wide types (though the first sub-circuit may be of either type).

We evaluate each sub-circuit $C_i$ using a method tailored to its type:

- **Shallow sub-circuits:** Apply our IC scheme (Theorem 6) to simulate the GMW evaluation of $C_i$. Since each layer has fewer than $\kappa$ gates, we have $|C_i| < \kappa \cdot \mathsf{depth}(C_i)$. The IC scheme achieves communication complexity $O_\varepsilon(n^2(|C_i| + \kappa))$ and round complexity $O_\varepsilon(|C_i| + \kappa) = O_\varepsilon(\kappa \cdot \mathsf{depth}(C_i) + \kappa)$.
- **Wide sub-circuits:** Evaluate using the standard GMW protocol with each message encoded via ECC of length $O_\varepsilon(\kappa + \log |C_i|)$. Since each layer has at least $\kappa$ gates, the GMW protocol for one layer involves $\Omega(n^2\kappa)$ bits of communication. The per-message ECC overhead is thus absorbed into this layer's communication, yielding communication complexity $O_\varepsilon(n^2|C_i|)$ and round complexity $O_\varepsilon(\mathsf{depth}(C_i))$.

In short, we use our proposed IC schemes to evaluate shallow circuits and standard ECC to evaluate wide circuits. This entire transformation yields a $O_\varepsilon(n^2(|C| + \kappa))$ communication cost MPC protocol over noisy channels with round complexity $O(\sum_{\text{odd } i}(|C_i| + \kappa) + \sum_{\text{even } i} \mathsf{depth}(C_i))$. Hence, the round complexity is bounded by

$$
\begin{aligned}
O\left(\sum_{\text{odd } i} \kappa \cdot (\mathsf{depth}(C_i) + \kappa) + \sum_{\text{even } i} \mathsf{depth}(C_i)\right) &= O\left(\kappa \cdot \left(\sum_i \mathsf{depth}(C_i) + \kappa\right)\right) \\
&= O(\kappa \cdot (\mathsf{depth}(C) + \kappa))
\end{aligned}
$$

*Other noisy channels.* While our IC scheme is designed for BSC channels, it can be readily extended to other natural noise models, including *elastic* and *erasure* channels, which we briefly define below.

An $(\alpha, \beta)$-elastic channel [DFMS04, Wul09] is similar to a BSC channel, where each bit sent on the channel is flipped with some fixed probability $\alpha$. However, it introduces an asymmetry: a corrupt receiver can increase the probability of correctly receiving the bit sent over the channel. This is formally captured by reducing the bit-flip probability to $\beta$ ($< \alpha$) for a corrupt receiver. Our proposed IC scheme and its analysis over BSC from Theorem 11 naturally extend to elastic networks given as a building block a constant-rate batch OT construction over elastic networks, which is known from [KMS16, CDLR16]. For more details, we point the reader to Theorem 21 in [GPZ18], which presents a similar extension from BSC to elastic networks in the context of two-party computation.

Another very natural model of noisy networks is an erasure channel, where each transmitted bit is either received correctly or erased (i.e., lost) with some fixed probability. Since erasure channels are strictly weaker than BSC channels, our construction also extends to this stochastic model. In fact, it is plausible that even simpler IC schemes may suffice in this setting. However, extending our approach to (mildly) adversarial noisy channels remains a challenging open problem, which we leave for future work.

# I Conclusions and Future Work

We initiate the study of information-theoretic MPC over noisy networks in the multiparty setting. We construct the first MPC protocol that remains secure against a dishonest majority of semi-honest parties while achieving optimal communication overhead in terms of circuit size: it evaluates any circuit $C$ with communication complexity $O_n(|C|)$. Our protocol departs from the classical MPC methodology by introducing a novel *interactive-coding* that defies channel noise by rewinding the protocol to a round prior to the corrupted transmission. Such rewinding gives rise to two significant challenges: privacy violations and inconsistent correlated randomness. Our protocol departs from the classical MPC methodology by incorporating a novel interactive coding scheme that handles channel noise by controlled rewinding the simulated execution to a point before a corrupted transmission. This rewinding introduces two central challenges: potential privacy violations and inconsistencies in the use of correlated randomness.

Our interactive coding scheme addresses both issues, enabling secure computation over noisy networks without increasing the asymptotic communication cost of the underlying computation. On the analytical side, our work provides the first treatment of multi-party interactive coding for randomized protocols in a noisy setting. We expect that the techniques developed here may be useful beyond MPC, particularly for distributed protocols in which randomness is adaptive and evolves throughout the execution, and under alternative models of channel noise.

Another natural direction is to extend our treatment to malicious adversaries. In this setting, one could compile a maliciously secure protocol using standard error-correcting codes to tolerate noisy channels: decoding errors merely result in arbitrary messages, which do not introduce new attacks thanks to the robustness of malicious security. In contrast, extending our interactive-coding approach to the malicious setting presents substantial challenges. Interactive coding relies on synchronization mechanisms that involve rewinding and careful coordination across parties; under malicious behavior, such mechanisms become significantly harder to analyze and secure, particularly when the protocol itself is randomized. Developing an interactive coding scheme that remains secure against malicious adversaries in the presence of noise is, therefore, a challenging and, in our view, promising direction for future research.