On the Round Complexity of Dishonest-Majority MPC*

Ran Cohen† Daniel Collins‡§ Pouyan Forghani§¶ Juan Garay‖ Vassilis Zikas¶**

September 15, 2026

Abstract

What is the round complexity of n-party secure multiparty computation (MPC) over point-to-point networks in the dishonest-majority setting with unanimous or identifiable abort?

Despite decades of research, this foundational question remains open. While two-round MPC is achievable given a broadcast channel, and broadcast protocols with expected-constant rounds exist tolerating any constant fraction of corruptions, a naïve combination of the two yields MPC with expected $O(\log n)$ rounds. The core obstacle lies in maintaining round efficiency under parallel composition without an honest majority. Moreover, existing lower bounds for standard broadcast do not apply to broadcast with abort, leaving the exact round complexity of this setting unknown.

In this work, we systematically investigate the round complexity of MPC over point-to-point networks, providing positive and negative results:

- We show that any broadcast protocol (and thus generic MPC) with unanimous abort tolerating super-constant corruptions requires super-constant rounds, ruling out strict-constant-round protocols.
- We construct a round-preserving, black-box parallel composition compiler for broadcast with unanimous abort. This yields our main result: Expected-constant-round MPC with unanimous abort in the PKI model against any constant fraction of corruptions, assuming oblivious transfer (OT) and verifiable random functions (VRFs).
- Finally, we prove that expected-constant round broadcast (and MPC) with identifiable abort is impossible against $n - o(n)$ corruptions.

*A preliminary version of this work appears in *Proc. TCC'26*.

†Efi Arazi School of Computer Science, Reichman University. E-mail: cohenran@runi.ac.il. Research supported in part by NSF grant no. 2055568 and by ISF grant 1834/23.

‡Department of Computer Science, New York University & School of Computer Science and Engineering, Hebrew University of Jerusalem. E-mail: dpc6906@nyu.edu.

§Part of this research was completed while affiliated with Texas A&M University.

¶School of Cybersecurity and Privacy, Georgia Institute of Technology. E-mail: {pforghani3,vzikas}@gatech.edu.

‖Department of Computer Science and Engineering, Texas A&M University. E-mail: garay@tamu.edu. Research supported in part by NSF Award CCF-2055694

**Research supported in part by the NSF Award No. 2448339, the Stellar Development Foundation, JPMorgan Chase, and Sunday Group, Inc., and done in part while the author was visiting the Simons Institute, UC Berkeley.

Contents

1 Introduction **1**
- 1.1 Our Contributions . . . 2
  - 1.1.1 Impossibility of Constant-Round UA-MPC . . . 2
  - 1.1.2 UA-MPC in Expected-Constant Rounds . . . 3
  - 1.1.3 Impossibility of Expected-Constant-Round IA-MPC . . . 5
- 1.2 Open Questions and Future Work . . . 6
- 1.3 Organization of the Paper . . . 6

2 Preliminaries and Model **6**
- 2.1 Security Model . . . 7
- 2.2 Primitives and Building Blocks . . . 8

3 Expected-Constant-Round UA-Parallel Broadcast **10**

4 Expected-Constant-Round UA-MPC Over P2P Channels **13**

5 Round-Complexity Lower Bounds for Broadcast with Abort **15**
- 5.1 Impossibility of Constant-Round UA-Broadcast . . . 15
- 5.2 Impossibility of Expected-Constant-Round IA-Broadcast . . . 16

Bibliography **19**

1 Introduction

Secure multiparty computation (MPC) [Yao86, GMW87] enables n mutually distrusting parties to jointly compute an arbitrary function over their inputs while guaranteeing *correctness* and *privacy* facing an adversary controlling up to t corrupted parties. In the dishonest-majority setting, which is the focus of this work, *fairness* cannot be achieved generically [Cle86], and standard security is relaxed to *security with abort*. Here, the adversary can first learn the output of the corrupted parties and subsequently decide whether to abort the protocol, preventing honest parties from receiving their outputs.

An important efficiency metric of MPC protocols is their round complexity, i.e., the maximal number of rounds required for every honest party to complete the protocol. This metric goes hand in hand with the underlying communication model. Starting with the seminal results from the '80s [GMW87, BGW88, CCD88, RB89], MPC protocols were initially designed in the *broadcast model*, where parties had access to a broadcast channel in addition to a fully connected network of point-to-point (P2P) channels. Once security is established in the broadcast model, Byzantine broadcast protocols [PSL80, LSP82] are used to instantiate the broadcast channel and obtain protocols over P2P channels. While this design principle preserves security, preserving the round complexity is far more subtle. Because MPC protocols typically invoke broadcast channels repeatedly and in parallel, maintaining round efficiency requires broadcast protocols that compose gracefully without round blowup.

Early deterministic broadcast protocols [PSL80, LSP82, DS83, BGP89, GM93] compose naturally, but as established by Dolev and Strong [DS83], their round complexity scales at least linearly with the number of corruptions ($t + 1$ rounds). To circumvent this lower bound, a long line of work initiated by Ben-Or [Ben83] and Rabin [Rab83] developed *randomized* broadcast protocols, culminating in constructions that terminate in expected-constant rounds [FM88, FG03, KK06].

Somewhat surprisingly though, plugging these protocols into broadcast-model MPC does not result in protocols over P2P with the same expected round complexity. As observed by Ben-Or and El-Yaniv [BE03], naïve parallel composition of n instances of such expected-constant-round protocols leads to expected round complexity of $O(\log n)$. Further, as highlighted by Lindell, Lysyanskaya, and Rabin [LLR02], since these randomized protocols cannot guarantee simultaneous termination of honest parties, sequential composition is also challenging: parties cannot synchronize the start of subsequent instances, and naïve synchronization strategies can lead to an exponential blowup in the round complexity.

Honest-majority MPC over P2P. In the honest-majority setting, where guaranteed output delivery is achievable, these challenges were successfully addressed in [BE03, LLR02, CCGZ19, CCGZ21], leading to round-preserving compilers from the broadcast model. In the information-theoretic setting, MPC over P2P channels was established in expected $O(d)$ rounds (where d is the multiplicative depth of the circuit) based on [BGW88, CCD88, RB89]. In the computational setting, expected $O(1)$-round MPC was achieved based on [BMR90, DI05].

Dishonest-majority MPC over P2P. In the dishonest-majority setting, where generic MPC necessitates security with abort, the landscape is more complex. Three distinct variants of security with abort are widely studied:

1. *Identifiable Abort (IA):* In case of an abort, all honest parties unanimously identify at least one corrupted party.

2. *Unanimous Abort (UA):* All honest parties detect an attack and abort together, but cannot necessarily identify the malicious party.

3. *Selective Abort (SA):* Honest parties do not necessarily agree on whether an attack occurred; some honest parties may receive output while others abort.

For the weaker security variant of selective abort, Goldwasser and Lindell [GL05] showed that every R-round MPC protocol in the broadcast model can be compiled into a $2R$-round protocol in the P2P model facing $t < n$ corruptions. Subsequently, building on [ACGJ18, ACGJ19], Cohen, Garay, and Zikas [CGZ20] showed that assuming two-round oblivious transfer (OT), every function can be computed with selective abort in just two P2P rounds for any $t < n$.

In contrast, little is known about the round complexity of MPC over P2P channels under the stronger, more desirable notions of unanimous or identifiable abort. Although expected-constant-round broadcast protocols exist in the dishonest-majority setting [GKKO07, FN09, WXSD25], they are not known to be composable in a round-preserving manner, as composition techniques developed for the honest-majority setting break down when $t \geq n/2$ (as elaborated on in Section 1.1). In fact, MPC with unanimous abort can be achieved in two rounds, where the first is over P2P and the second over broadcast [CGZ20]; however, as every party broadcasts in the second round, instantiating the broadcast round with state-of-the-art expected-constant-round broadcast protocols (e.g., [WXSD25]) yields an expected round complexity of $O(\log n)$ due to parallel composition.

Furthermore, existing round lower bounds on broadcast protocols (e.g., [DS83, CMS89, KY84, GKKO07, CPS20, CHM+22]) consider security with guaranteed output delivery and do not directly apply to the broadcast protocols that permit unanimous abort. This leads us to the following fundamental questions:

> *What is the round complexity of MPC over point-to-point channels with unanimous or identifiable abort in the dishonest-majority setting?*

And, more specifically:

> *Is (expected-)constant-round MPC over P2P channels with unanimous or identifiable abort at all possible in the dishonest-majority setting?*

1.1 Our Contributions

In this work, we systematically investigate these questions and provide both positive and negative results.

1.1.1 Impossibility of Constant-Round UA-MPC

Our first result shows that strict constant-round generic MPC with unanimous abort facing a super-constant number of corruptions is impossible over P2P networks. In fact, this result holds for the privacy-free primitive of broadcast [PSL80, LSP82]. A t-secure broadcast protocol allows a designated sender holding input $x \in \{0,1\}$ to distribute x to n parties while satisfying *consistency* (all honest parties agree on an output) and *validity* (if the sender is honest, honest parties output x). Initial lower bounds for t-secure broadcast considered error-free deterministic protocols and established the necessity of $t+1$ rounds [FL82, DLM82, DS83, Du89]. Based on these results, lower bounds for randomized protocols that permit a negligible error were devised, showing the impossibility of constant-round broadcast facing $t = \omega(1)$ corruptions [CMS89, KY84]. These results directly imply impossibility for constant-round MPC with guaranteed output delivery over P2P channels.

We show that the impossibility proofs of [CMS89, KY84] extend even to broadcast with **unanimous abort**, where validity is relaxed so that with overwhelming probability the common output is either the sender's input or a special abort symbol $\perp$ (with $\perp$ allowed only when at least one party is corrupted).

The high-level idea is to assume, towards a contradiction, that there exists an r-round, t-secure broadcast with abort protocol for $t \geq r$. In an all-honest execution where the sender holds input 0, the common output must be 0. Following a hybrid sequence of scenarios similar to [CMS89, KY84], where at most t parties are corrupted in each scenario and neighboring scenarios share an honest party whose view is identically distributed, we establish that no scenario aborts except with negligible probability. These scenarios enable replacing the sender's input from 0 to 1 (as, in particular, no party ever aborts except with negligible probability), resulting in an all-honest execution in which the sender begins with input 1 but parties output 0 with noticeable probability, yielding a contradiction.

1.1.2 UA-MPC in Expected-Constant Rounds

Having ruled out strict constant-round UA-MPC, we proceed to prove the next best guarantee: MPC with UA in *expected* constant rounds is achievable. We construct such a protocol assuming oblivious transfer (OT) and verifiable random functions (VRFs) in the PKI model for signatures and VRFs. In this PKI model, (which is inherited from [WXSD25]), honest parties correctly generate their signatures keys and VRFs keys, whereas corrupted parties may maliciously choose their signatures keys (even after seeing honest parties' public keys) but have their VRF keys sampled honestly via trusted setup. All parties hold their own secret keys and the public keys of all participating parties at the onset of the protocol.

Before describing our result, we discuss the challenges that arise when using expected-constant-round broadcast for realizing a broadcast channel in MPC protocols. In such a setting, the broadcast protocol must support both *parallel composition* since every party may speak in a given broadcast round, and *sequential composition* since broadcast rounds may occur one after the other.

Challenges of parallel composition of broadcast. As observed by Ben-Or and El-Yaniv [BE03] (see also [CCGZ19]), executing in parallel n instances of expected-constant-round broadcast to realize parallel broadcast may result in a protocol with expected $O(\log n)$ rounds. Ben-Or and El-Yaniv proposed an approach for preserving the round complexity of parallel broadcast in the honest-majority setting. Their approach relies on Byzantine Agreement (BA) which is a primitive that enables parties to agree on one of the honest parties' input bits. The high-level idea works as follows:

- The parties execute m copies of each of the n instances (for $m \approx \log n$), but for a fixed number of rounds, such that with high probability at least one instance from each batch will complete.
- Upon completion, each party checks for each of the n batches if it received output in at least one of the executions. If so, it arbitrarily selects one output for each batch and distributes the vector of output values to all the parties.
- Next, the parties run a leader-election protocol (which internally uses BA). Once some party P_k is elected to be the leader, all parties run a BA protocol on the output vector that was previously distributed by the leader P_k (which might be null).
  Each party checks whether the output of P_k corresponds to the output values it obtained in the first step and sets a termination indicator accordingly.
- Finally, the parties run another BA protocol on the termination indicators. If the output is 1 they terminate; otherwise, another iteration is executed.

Each iteration takes expected-constant rounds, and conditioned on the event that the leader is honest and obtained an output value in each of the n batches, the parties successfully reach agreement and terminate. For the right choice of parameters, this event occurs with constant probability, meaning that the protocols terminates after a constant number of iterations, in expectation.

The main obstacle in extending this approach (which was also used in [CCGZ19, CCGZ21, CFG+23]) to the dishonest-majority setting is the reliance on BA. While the use of BA in the leader-election protocol can be potentially replaced by a VRF, the need to agree on the leader's values and on termination is more challenging. The problem is that standard BA cannot be realized without an honest majority [Fit03]. Indeed, by definition, an adversary controlling a majority of the parties can force the output to any value of its choice. For example, the adversary can systematically prevent termination in the last BA of each iterations just by choosing its inputs to "no termination."

Challenges of sequential composition of broadcast. Sequential composition introduces a separate obstacle as randomized broadcast protocols cannot guarantee simultaneous termination among honest parties. As observed by Lindell, Lysyanskaya, and Rabin [LLR02], a party that completes an instance does not know whether it is a fast party (and others are still running) or a slow party (and everyone else has already finished). In case the gap between fast and slow parties can be c rounds, one approach is to execute every "protocol round" over $2c + 1$ communication rounds in the following manner: every message that arrives within this window is counted for the protocol round, and the party will send its actual message in round $c+1$. This approach ensures that although parties may be locally unsynchronized, they proceed in a globally synchronized manner. The downside of this approach is that only a constant number of instances may be supported before the blowup is super-constant (and a super-logarithmic number of instances results in a super-polynomial blowup).

A more efficient approach used in the honest-majority setting is to reduce the gap between fast and slow parties to a single round using the protocol proposed by Bracha [Bra84]. Here, once a obtaining the output, each party sends the output value to all parties and keeps running. Once obtaining $t + 1$ termination messages on a given value, a party can accept this value as its output message (even if it did not obtain an output from the protocol just yet) and sends this value to all parties. Once obtaining $n-t$ termination messages on a given value, the party can safely terminate, knowing that all honest parties will also terminate by the next round. This approach, however, is inherently limited to the honest-majority setting and does not extend to the dishonest-majority realm.

Our construction. We overcome these obstacles by designing a black-box parallel composition compiler for broadcast that offers security with unanimous abort. Given any R-round UA-MPC protocol π in the broadcast model (for $R = O(\log n)$), our compiler outputs a UA-MPC protocol over P2P running in expected $O(R \cdot 2^R)$ rounds. Combined with a constant-round broadcast-model MPC protocol and the expected-constant-round broadcast protocol of [WXSD25], this yields an expected-constant-round UA-MPC protocol over P2P channels against any constant fraction of corruptions.

In more detail, our construction consists of the following steps:

1. **Reactive Parallel Broadcast (RPBC) Functionality:** We model the broadcast channel in π as a reactive ideal functionality for *parallel broadcast* (i.e., *interactive consistency* [PSL80]), where in every round every party can send a message to the broadcast functionality, and each party obtains a vector of messages sent by all parties.

   We assume that all communication in π is over broadcast and that P2P channels are not

used; this is without loss of generality, since similarly to the pre-compiler in [GMW87], every message over P2P can be encrypted using a public-key encryption scheme and sent over the broadcast channel.

The protocol π may use multiple calls for parallel broadcast, and we wish to capture this via sequential executions of a parallel-broadcast protocol using a *single* instance of the PKI. Similarly to [CL17], we model this using a *reactive* functionality for parallel broadcast (RPBC).

2. **Relaxation to Broadcast with UA:** We relax the ideal functionality for RPBC to provide security with unanimous abort. That is, once honest parties send their inputs in a given round, the adversary is informed of these values and can choose to either provide messages for the corrupted parties or send an abort message. In the latter case, the RPBC functionality returns $\perp$ to all honest parties in this call and in all subsequent calls.

3. **Black-Box Parallel Broadcast Compiler:** Given a t-secure broadcast protocol, we construct—in a black-box manner—a parallel-broadcast protocol that is t-secure with unanimous abort. This construction assumes a PKI for signatures, as well as the setup needed for the underlying broadcast protocol. The main idea is inspired by techniques used for information-theoretic, semi-honest, honest-majority MPC [DN07] and applies them in the computational setting and where the malicious parties constitute a majority.

   In each call for parallel broadcast, every party P_i, signs its input x_i as $\sigma_i \leftarrow \mathsf{Sign}_{\mathsf{sk}_i}(x_i)$ and sends (x_i, σ_i) to a designated aggregator P_1 over a P2P channel. Next, P_1 broadcasts the concatenated vector $((x_1, \sigma_1), \ldots, (x_n, \sigma_n))$ using a single instance of a t-secure broadcast protocol. Finally, each party locally verifies all signatures in the output vector: in case they all pass verification the party accepts these messages, but if any signature fails to verify, the party aborts.

   Because all parties obtain the same broadcast output, any invalid signature causes all honest parties to abort simultaneously, ensuring unanimous abort. Further, the security of the signature scheme prevents a cheating P_1 from altering honest parties' messages. Finally, we note that in case of an abort, i.e., when a signature of some P_j does not verify, the parties cannot identify which party is cheating: P_j or P_1; therefore, this approach does not yield security with *identifiable* abort. As a result, the underlying broadcast protocol does not need to guarantee output delivery or identify cheaters; instead, broadcast that is UA-secure can be used.

   Although we assume that all parties begin the first call simultaneously, as the underlying broadcast protocol cannot guarantee simultaneous termination, and as we cannot apply the termination technique from [Bra84] (which relies on an honest majority), we use the round expansion idea described above to support sequential composition. This is the source of the exponential term in the round complexity. Nonetheless, once we apply this compiler with constant-round MPC (e.g., the OT-based two-round MPC from [GS18, BL18]) and with the broadcast protocol from [WXSD25], we obtain our desired result: Expected-constant-round UA-MPC over P2P channels.

1.1.3 Impossibility of Expected-Constant-Round IA-MPC

After establishing the feasibility of expected-constant-round UA-MPC facing a constant fraction of corruptions, we proceed to see if a similar result can be obtained facing more corruptions, i.e., for $t = n - o(n)$. Towards this goal, we revisit the lower bound from [GKKO07], who showed that broadcast with guaranteed output delivery necessitates $\Omega(n/(n-t))$ rounds.

To illustrate the challenges of this question, we consider the result from [GKKO07] for $t = n-2$.

Their idea was to construct a chain where P_i is connected to P_{i-1} and P_{i+1} for every $1 < i < n$ (see Figure 3 in Section 5.2). The view of every pair of neighboring parties on the chain is distributed as in an execution where all other parties are cheating; hence, all neighboring pairs must output the same value, and so *all* parties must output the same value. Since there are $n-1$ hops in the chain, if the protocol completes in $n-2$ rounds, P_n will not be able to distinguish if the broadcaster P_1 has input 0 or 1, and a contradiction is easily derived.

This lower bound crucially relies on the robustness of the broadcast protocol. Indeed, for protocols secure with *unanimous abort*, this attack will not yield a contradiction, since every party can sense the attack, and so all parties can unanimously abort regardless of the sender's input.

Our next result shows that for broadcast that is secure with *identifiable* abort, the $\Omega(n/(n-t))$ lower bound still applies. The high-level idea is that pairwise agreement in the chain described above implies that if any party identifies a cheater, say P_{i^*}, then all parties identify P_{i^*} as cheater; in particular, P_{i^*} will identify itself as the cheater in an execution where it plays honestly. Therefore, no party can be identified as a cheater and the output must be a binary value, which in turn enables the lower bound from [GKKO07] to kick in. This means that catching cheaters in broadcast protocols is as expensive (in term of round complexity) as preventing cheating altogether.

1.2 Open Questions and Future Work

In this work we initiate a comprehensive study of the round complexity of MPC in the dishonest-majority setting over P2P channels, establishing the feasibility of expected-constant-round UA-MPC, together with barriers for constant-round UA-MPC and expected-constant-round IA-MPC. These results point us to many interesting follow-up questions, such as:

- Is it possible to obtain UA-MPC with expected-constant rounds in the presence of $n - o(n)$ corruptions, or does a matching lower bound exist?
- Is there a dishonest-majority analogue of Bracha's termination protocol [Bra84] that prevents round blowup during sequential composition without simultaneous termination?
- Can the PKI setup assumption that requires honestly generated VRF keys for the broadcast protocol from [WXSD25] be relaxed to standard PKI setup?

1.3 Organization of the Paper

The rest of the paper is organized as follows. Preliminaries and security definitions are presented in Section 2. We then present our positive results: the round-preserving compiler for parallel broadcast with UA is described in Section 3, and its application to UA-MPC in Section 4. Finally, we present our impossibility results in Section 5: the impossibility of constant-round UA-broadcast, followed by the impossibility of expected-constant-round IA-broadcast.

2 Preliminaries and Model

Notation. We denote the security parameter by κ. For $n \in \mathbb{N}$, let $[n] = \{1, \ldots, n\}$. A function $\mu : \mathbb{N} \to [0,1]$ is *negligible* if $\mu(\kappa) < 1/p(\kappa)$ for every positive polynomial p and all sufficiently large κ. We write PPT for probabilistic polynomial-time algorithms and use $\approx_c$ to denote computational indistinguishability. For a probability distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$ to denote sampling x according to $\mathcal{D}$.

2.1 Security Model

Real/ideal simulation paradigm. We consider a standard real/ideal simulation-based definition of secure multiparty computation; see [GMW87, Can00]. For simplicity, we present definitions in the stand-alone setting with static security in which the set of corrupted parties $I \subset [n]$ is fixed prior to protocol execution. Our feasibility results extend naturally to adaptive security. We follow the terminology from [CGZ20] and begin by defining secure function evaluation with unanimous and identifiable abort followed by respective broadcast primitives.

Real-world execution. An n-party protocol $\pi = (P_1, \ldots, P_n)$ is an n-tuple of interactive PPT Turing machines. The term P_i refers to the i^{th} interactive machine. Each party P_i starts with an input $x_i \in \{0,1\}^*$ and uniformly distributed random coins $r_i \in \{0,1\}^*$. Without loss of generality, we assume that the input length is bounded by the security parameter κ. An adversary $\mathcal{A}$ is another interactive PPT machine representing the corrupted parties. It starts the execution with input consisting of the identities of the corrupted parties, their private inputs, and an auxiliary input.

The parties execute the protocol in a synchronous network, where the execution proceeds in rounds. Each round consists of a send phase followed by a receive phase. Messages sent in round r are guaranteed to be delivered before round $r+1$ begins. The adversary is assumed to be *rushing*: in any round, it observes all messages sent by honest parties before determining the messages sent by corrupted parties. Parties communicate over a complete network of point-to-point authenticated channels; when considering the broadcast model, every party can also broadcast a message to all other parties.

Honest parties follow the protocol specifications. Corrupted parties are controlled by $\mathcal{A}$ and may deviate arbitrarily. At the conclusion of the execution, each honest party outputs its prescribed output, while the adversary outputs its view of the computation, consisting of the inputs, random coins, and all messages observed by the corrupted parties.

Definition 2.1 (Real-world execution)**.** *Let $\pi = (P_1, \ldots, P_n)$ be an n-party protocol and let $I \subseteq [n]$ denote the set of parties corrupted by $\mathcal{A}$. The real-world execution of π with adversary $\mathcal{A}$ on input vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, auxiliary input z, and security parameter κ, denoted by $\text{REAL}_{\pi,\mathcal{A}(z)}(\boldsymbol{x}, \kappa)$, is the joint output distribution of $P_1, \ldots, P_n$ and $\mathcal{A}(z)$ resulting from the above protocol interaction.*

Ideal-world execution with abort. We consider the standard formulation of ideal-world execution with abort (see [GL05, IOZ14, CL17]). Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-ary function. Let $\mathcal{S}$ be a PPT simulator controlling a subset $I \subseteq [n]$ of corrupted parties. The ideal execution of f on input vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, auxiliary input z, and security parameter κ proceeds as follows.

- **Sending inputs to the trusted party.** Each honest party P_i, for $i \notin I$, sends its input x_i to the trusted party. The simulator $\mathcal{S}$ sends arbitrary inputs x_i' on behalf of corrupted parties. Let x_i' denote the input provided for party P_i.
- **Trusted party answers the simulator.** The trusted party computes $(y_1, \ldots, y_n) := f(x_1', \ldots, x_n')$ and sends y_i to $\mathcal{S}$ for every $i \in I$.
- **Simulator response.** $\mathcal{S}$ responds with either `continue` or `abort`.
- **Trusted party answers parties.** If $\mathcal{S}$ sent `abort` and $I \neq \emptyset$, the trusted party sets $y_j := \perp$ for every $j \notin I$. Next (in any case, `continue` or `abort`), send y_j to P_j for every $j \notin I$.
- **Outputs.** Each honest party outputs the value received from the trusted party. $\mathcal{S}$ outputs an arbitrary function of its auxiliary input, the inputs of the corrupted parties, and the messages it received from the trusted party.

Definition 2.2 (Ideal-world execution with unanimous abort)**.** *The joint output distribution of the above ideal execution with simulator $\mathcal{S}$, corrupted set I, input vector $\boldsymbol{x}$, auxiliary input z, and security parameter κ is denoted by* $\text{IDEAL}^{\mathsf{ua}}_{f,I,\mathcal{S}(z)}(\boldsymbol{x},\kappa)$.

We define the following variant of security with identifiable abort.

- **Ideal execution with identifiable abort.** This ideal execution proceeds as above, except that to abort, $\mathcal{S}$ sends $(\texttt{abort}, i^*)$ for some $i^* \in I$. In this case, the trusted party sends $(\perp, i^*)$ to every honest party. The resulting joint output distribution is denoted $\text{IDEAL}^{\mathsf{ia}}_{f,I,\mathcal{S}(z)}(\boldsymbol{x},\kappa)$.

Security definition. Having defined the real and ideal computation, we can now define the security of protocols.

Definition 2.3 (Security)**.** *Let* $\mathsf{type} \in \{\mathsf{ua}, \mathsf{ia}\}$ *and let* $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ *be an n-ary function. A protocol π t-securely computes f with* type *if for every PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{S}$ such that for every $I \subseteq [n]$ of size at most t, it holds that*

$$\left\{\text{REAL}_{\pi,I,\mathcal{A}(z)}(\boldsymbol{x},\kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\,\kappa\in\mathbb{N}} \approx_c \left\{\text{IDEAL}^{\mathsf{type}}_{f,I,\mathcal{S}(z)}(\boldsymbol{x},\kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\,\kappa\in\mathbb{N}}.$$

For simplicity of notation, we often prefix a primitive by its abort type. Thus, UA-MPC and IA-MPC denote MPC with unanimous and identifiable abort, respectively. We use the same convention for the broadcast primitives introduced below; for example, UA-broadcast and IA-broadcast denote broadcast with unanimous abort and identifiable abort, respectively.

2.2 Primitives and Building Blocks

We formulate ideal functionalities for single-shot broadcast, parallel broadcast, and reactive parallel broadcast supporting sequential calls. As we focus on static corruptions, we do not consider "corruption-fair" broadcast functionalities [HZ10, GKKZ11, CGZ23], in which an adaptive adversary cannot first learn an honest sender's message and later corrupt the sender and change its message. Nonetheless, our modeling can be adjusted to capture this property using the approach of [CGZ23].

Although one can model broadcast as a special case of secure function evaluation leaking the sender's input to the adversary (as done, e.g., in [CCGZ19]), we present an explicit ideal model for broadcast, to simplify the various notions we capture.

Broadcast. We first present the ideal model for single-shot broadcast with unanimous abort; let $s \in [n]$ denote the index of the designated sender.

- **Sending inputs to the trusted party.** An honest sender P_s sends its input $x \in \{0,1\}^*$ to the trusted party, who forwards x to the $\mathcal{S}$. If P_s is corrupted, $\mathcal{S}$ sends an arbitrary value x on behalf of P_s .
- **Simulator response.** $\mathcal{S}$ either sends `continue` or sends `abort`.
- **Trusted party answers parties.** If $\mathcal{S}$ sent `abort` (and $I \neq \emptyset$), the trusted party sends $\perp$ to every honest party P_j with $j \notin I$. Otherwise, the trusted party sends x to every P_j with $j \notin I$.
- **Outputs.** Every honest party outputs the value received from the trusted party and the simulator outputs an arbitrary function of its view.

Definition 2.4 (Broadcast with unanimous abort)**.** *The joint output distribution of the above ideal execution with simulator $\mathcal{S}$, corrupted set I, input value x_s, auxiliary input z, and security parameter κ is denoted by* $\text{IDEAL}^{\mathsf{ua\text{-}bc}}_{I,\mathcal{S}(z)}(\boldsymbol{x},\kappa)$.

We proceed to define security with guaranteed output delivery and with identifiable abort.

- **Broadcast with guaranteed output delivery.** This execution proceeds as in the unanimous-abort ideal execution, except that the adversary cannot abort the computation. We denote the resulting distribution by $\text{IDEAL}^{\mathsf{gd\text{-}bc}}_{I,\mathcal{S}(z)}(x_s, \kappa)$.
- **Broadcast with identifiable abort.** This execution proceeds as in the unanimous-abort ideal execution, except that to abort the computation, $\mathcal{S}$ sends $(\texttt{abort}, i^*)$ for some $i^* \in I$. In this case, the trusted party sends $(\perp, i^*)$ to every honest party. We denote the resulting distribution by $\text{IDEAL}^{\mathsf{ia\text{-}bc}}_{I,\mathcal{S}(z)}(x_s, \kappa)$.

Definition 2.5 (Security)**.** *Let* $\mathsf{type} \in \{\mathsf{gd}, \mathsf{ua}, \mathsf{ia}\}$*. A protocol* π*, in which* P_s *holds input* x_s *and all other parties have no input, is a* t*-secure broadcast with* type *if for every PPT adversary* $\mathcal{A}$*, there exists a PPT adversary* $\mathcal{S}$ *such that for every* $I \subseteq [n]$ *of size at most* t*, it holds that*

$$\left\{\text{REAL}_{\pi,I,\mathcal{A}(z)}(\boldsymbol{x}, \kappa)\right\}_{(x_s,z)\in(\{0,1\}^*)^{n+1},\, \kappa\in\mathbb{N}} \approx_c \left\{\text{IDEAL}^{\mathsf{type\text{-}bc}}_{I,\mathcal{S}(z)}(\boldsymbol{x}, \kappa)\right\}_{(x_s,z)\in(\{0,1\}^*)^{n+1},\, \kappa\in\mathbb{N}}.$$

Assuming the existence of EUF-CMA signatures and VRFs, together with a PKI setup for their verification keys, the protocol of Wan *et al.* [WXSD25] realizes broadcast with guaranteed output delivery in expected constant rounds tolerating any constant fraction of corruptions.

Parallel broadcast. Parallel broadcast (interactive consistency [PSL80]) requires every party to act as a sender simultaneously. As in the single-shot case, we present the ideal model for parallel broadcast with unanimous abort; security with identifiable abort and with guaranteed output delivery is similarly defined.

- **Sending inputs to the trusted party.** Each honest party P_i, for $i \notin I$, sends its input $x_i \in \{0,1\}^*$ to the trusted party, who forwards x_i to the simulator. $\mathcal{S}$ sends arbitrary inputs x'_i on behalf of corrupted parties. Let x'_i denote the input provided for party P_i.
- **Simulator response.** $\mathcal{S}$ either sends `continue` or sends `abort`.
- **Trusted party answers parties.** If $\mathcal{S}$ sent `abort` and $I \neq \emptyset$, the trusted party sends $\perp$ to P_j for every $j \notin I$. Otherwise, the trusted party send $(x'_1, \ldots, x'_n)$ to P_j for every $j \notin I$.
- **Outputs.** Each honest party outputs the value received from the trusted party and $\mathcal{S}$ outputs an arbitrary function of its view.

For $\mathsf{type} \in \{\mathsf{gd}, \mathsf{ua}, \mathsf{ia}\}$ we denote the ideal computation for parallel broadcast with type by $\text{IDEAL}^{\mathsf{type\text{-}pbc}}_{I,\mathcal{S}(z)}(\boldsymbol{x}, \kappa)$, and say that a protocol π is a t-secure parallel broadcast with type if for every PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{S}$ such that for every $I \subseteq [n]$ of size at most t, it holds that

$$\left\{\text{REAL}_{\pi,I,\mathcal{A}(z)}(\boldsymbol{x}, \kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\, \kappa\in\mathbb{N}} \approx_c \left\{\text{IDEAL}^{\mathsf{type\text{-}pbc}}_{I,\mathcal{S}(z)}(\boldsymbol{x}, \kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\, \kappa\in\mathbb{N}}.$$

Reactive functionality for parallel broadcast. To support sequential invocations as needed in MPC protocols, we define a reactive parallel broadcast functionality. Here, the trusted party can maintain across sequential invocations state indicating whether an abort occurred. Each invocation proceeds as in the ideal model for parallel broadcast defined above, with the exception that in case the simulator sent `abort` in a certain invocation, the trusted party returns $\perp$ to all honest parties in every future invocation. We note that when instantiating such a reactive functionality for broadcast, the protocol can use its existing PKI setup for the multiple instances that are invoked sequentially.

Digital signatures. A signature scheme consists of three PPT algorithms. The key-generation algorithm $\mathsf{Gen}(1^\kappa)$ outputs a key pair $(\mathsf{sk}, \mathsf{pk})$. The signing algorithm $\mathsf{Sign}_{\mathsf{sk}}(m)$ on a message m produces a signature σ, and the verification algorithm $\mathsf{Ver}_{\mathsf{pk}}(m, \sigma)$ outputs a bit indicating if σ is a valid signature on m.

We assume existential unforgeability under chosen-message attacks (EUF-CMA) [GMR88]. That is, no PPT adversary can forge a valid signature on a new message with non-negligible probability, even after adaptively querying signatures on messages of its choice. In our protocols, each party P_i is equipped with a key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ generated as part of the correlated randomness, and all verification keys are known to all parties. As out usage of signatures is black-box, we can use the formulation of [CDL+25].

Verifiable random functions. Some of our instantiations assume a secure VRF scheme $\mathsf{VRF} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Ver})$ [MRV99]. The key-generation algorithm $\mathsf{Gen}(1^\kappa)$ outputs a VRF key pair $(\mathsf{vsk}, \mathsf{vpk})$. The evaluation algorithm $\mathsf{Eval}(\alpha)$ takes an input α and outputs a value-proof pair (y, π). The verification algorithm $\mathsf{Ver}(\mathsf{vpk}, \alpha, y, \pi)$ outputs a bit indicating if (y, π) is a valid VRF evaluation for input α under vpk. We assume standard VRF guarantees: Uniqueness of valid outputs and pseudorandomness of y on inputs not queried to the evaluation oracle.

Oblivious transfer. Some of our MPC instantiations assume a maliciously secure OT protocol. OT is a two-party primitive between a sender holding a pair of messages (m_0, m_1) and a receiver holding a choice bit b. The receiver learns only m_b while the sender learns nothing about b.

Public-key infrastructure. We consider two variants of PKI: for signatures and VRFs. A PKI for signatures registers a signature verification key pk_i for every party P_i that is distributed to all protocol participants. An honest P_i honestly generates $(\mathsf{sk}_i, \mathsf{pk}_i)$ and keeps sk_i private. The adversary may arbitrarily choose corrupted parties' keys after seeing all the public keys for the honest parties.

A PKI for VRFs is stronger. All VRF key pairs $(\mathsf{vsk}_i, \mathsf{vpk}_i)$, including those of corrupted parties, are generated honestly by the setup. Each party obtains its own secret key vsk_i and the public keys of all parties $(\mathsf{vpk}_1, \ldots, \mathsf{vpk}_n)$. This distinction is important because VRF keys determine public randomness and therefore must be honestly generated even for corrupted parties to avoid grinding attacks.

3 Expected-Constant-Round UA-Parallel Broadcast

In this section, we present a compiler, denoted $\mathsf{Comp}_{\mathsf{ua\text{-}rpbc}}$, that transforms any single-sender broadcast protocol into a reactive parallel broadcast protocol with unanimous abort. The compiler asymptotically preserves the expected round complexity for a logarithmic number of sequential invocations.

We assume that the underlying single-sender broadcast protocol is secure under sequential composition using the same reusable setup, where each invocation is distinguished by a fresh session identifier [LLR06]. Standard PKI-based broadcast protocols, such as the expected-constant-round protocol of Wan *et al.* [WXSD25], naturally satisfy this composition requirement. For generality, we formulate the compiler assuming the underlying single-sender protocol guarantees unanimous abort (UA).

Handling non-simultaneous termination. Broadcast protocols with expected constant rounds cannot maintain simultaneous termination; that is, different parties may terminate in different rounds. A protocol has a *termination gap of at most c* if, whenever an honest party terminates in round r, every honest party terminates by round $r + c$. In the case of unanimous abort, this condition means that if some honest party aborts in round r, then every honest party aborts by round $r + c$.

This creates a challenge for sequential executions, since if honest parties terminate session s in different rounds, then they will also start session $s + 1$ in different rounds, and will no longer be synchronized. To preserve expected round complexity while accommodating non-simultaneous start, we employ the round-expansion technique of Lindell, Lysyanskaya, and Rabin [LLR02] (also used in [CCGZ19]). Each protocol round is expanded into $2c + 1$ communication rounds. During this window, parties listen for messages expected for the corresponding protocol round, and send their own message in round $c+1$. For simplicity, we state the compiler assuming that the underlying single-sender broadcast protocol has a termination gap 1.

Let Δ_s denote the termination gap after the first s sequential sessions. Initially, $\Delta_0 = 0$. Since the only non-simultaneous part of $\mathsf{Comp}_{\mathsf{ua\text{-}rpbc}}(\pi_{\mathsf{ua\text{-}bc}})$ is the invocation of $\pi_{\mathsf{ua\text{-}bc}}$, the single-session parallel-broadcast compiler also has a termination gap at most one protocol round. Therefore, if session $s + 1$ is run with expansion factor $2\Delta_s + 1$, then the resulting termination gap satisfies $\Delta_{s+1} = 2\Delta_s + 1$. Solving this recurrence gives $\Delta_s = \sum_{j=0}^{s-1} 2^j = 2^s - 1$.

Compiler $\mathsf{Comp}_{\mathsf{ua\text{-}rpbc}}(\pi_{\mathsf{ua\text{-}bc}})$

Inputs: In each session $s \in \mathbb{N}$, every party P_i receives input $x_i^s \in \{0,1\}^*$.

Setup phase: A PKI for a signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$, establishing key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ for party P_i. The setup for the underlying protocol $\pi_{\mathsf{ua\text{-}bc}}$ is generated by running its setup phase. Each party then receives its share of the setup at the onset of the execution.

Expanded-round convention: In session s, every logical protocol round is executed over $2^s - 1$ communication rounds. Parties record all incoming messages during the window and transmit their outgoing messages during sub-round 2^{s-1}.

Protocol execution for session s:

- Each party P_i signs its input as $\sigma_i^s = \mathsf{Sign}_{\mathsf{sk}_i}(s, i, x_i^s)$ and sends $(s, i, x_i^s, \sigma_i^s)$ to P_1.
- Party P_1 forms a vector $\boldsymbol{m}^s := (m_1^s, \ldots, m_n^s)$, setting m_i^s to the message received from P_i (or a default value if no valid message arrived). Party P_1 invokes $\pi_{\mathsf{ua\text{-}bc}}$ as sender with session identifier s and input $\boldsymbol{m}^s$.
- If $\pi_{\mathsf{ua\text{-}bc}}$ terminates with $\perp$, each party outputs $\perp$ for session s and all subsequent sessions. Otherwise, given output $\tilde{\boldsymbol{m}}^s = (\tilde{m}_1^s, \ldots, \tilde{m}_n^s)$, each party verifies that every entry $\tilde{m}_i^s$ parses as $(s, i, \tilde{x}_i^s, \tilde{\sigma}_i^s)$ and satisfies $\mathsf{Ver}_{\mathsf{pk}_i}((s, i, \tilde{x}_i^s), \tilde{\sigma}_i^s) = 1$. If any verification fails, output $\perp$ for session s and all subsequent sessions; otherwise, output $(\tilde{x}_1^s, \ldots, \tilde{x}_n^s)$ for session s.

Figure 1: Compiling single-sender UA-broadcast to reactive UA-parallel-broadcast over point-to-point channels.

Theorem 3.1. *Assume that EUF-CMA signatures exist and let $\pi_{\mathsf{ua\text{-}bc}}$ be a t-secure single-sender UA-broadcast protocol with expected R rounds, termination gap 1, and security under sequential composition using the same setup.*

Then $\pi_{\mathsf{ua\text{-}rpbc}} := \mathsf{Comp}_{\mathsf{ua\text{-}rpbc}}(\pi_{\mathsf{ua\text{-}bc}})$ is a t-secure reactive UA-parallel-broadcast protocol for $O(\log n)$ sequential calls in the PKI model for signatures. The expected round complexity of the s^{th} sequential call is $(2^s - 1)(R + 1)$.

Proof. Let $\mathcal{A}$ be a real-world PPT adversary interacting with $\pi_{\mathsf{ua\text{-}rpbc}}$; denote by $\mathcal{I} \subseteq [n]$ of size $|\mathcal{I}| \leq t$ the set of indices of corrupted parties. We construct a simulator $\mathcal{S}$ interacting with the ideal reactive parallel-broadcast functionality with unanimous abort such that the real and ideal executions are indistinguishable.

Simulator. The simulator $\mathcal{S}$ invokes $\mathcal{A}$ and emulates the PKI and the setup for $\pi_{\mathsf{ua\text{-}bc}}$ as in the real execution. $\mathcal{S}$ generates honest key pairs $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{Gen}(1^\kappa)$ for $i \notin \mathcal{I}$, reveals $\{\mathsf{pk}_i\}_{i \notin \mathcal{I}}$ to $\mathcal{A}$, and receives $\{\mathsf{pk}_j\}_{j \in \mathcal{I}}$ from $\mathcal{A}$.

For each session s, the simulator $\mathcal{S}$ proceeds as follows:

- **Simulating input aggregation:** Upon receiving honest inputs $\{x_i^s\}_{i \notin \mathcal{I}}$ from the functionality, $\mathcal{S}$ computes $\sigma_i^s \leftarrow \mathsf{Sign}_{\mathsf{sk}_i}(s, i, x_i^s)$ for every $i \notin \mathcal{I}$.
  - If P_1 is honest, for every $i \notin \mathcal{I}$ simulate sending $(s, i, x_i^s, \sigma_i^s)$ to P_1 and send $(s, i, x_i^s, \sigma_i^s)$ to $\mathcal{A}$ as leakage of the authenticated channels. Next, for every $i \in \mathcal{I}$ obtain from $\mathcal{A}$ the message $(s, i, x_i^s, \sigma_i^s)$.
  - If P_1 is corrupted, for every $i \notin \mathcal{I}$ send $(s, i, x_i^s, \sigma_i^s)$ to $\mathcal{A}$.
- **Simulating broadcast protocol:** The execution of the underlying broadcast protocol $\pi_{\mathsf{ua\text{-}bc}}$ defines an induced adversary $\mathcal{A}^s_{\mathsf{ua\text{-}bc}}$ for the s^{th} invocation of $\pi_{\mathsf{ua\text{-}bc}}$. By the security of $\pi_{\mathsf{ua\text{-}bc}}$, there exists a simulator $\mathcal{S}^s_{\mathsf{ua\text{-}bc}}$ for $\mathcal{A}^s_{\mathsf{ua\text{-}bc}}$.
  - If P_1 is honest, $\mathcal{S}$ sets the vector $\boldsymbol{m}^s = (m_1^s, \ldots, m_n^s)$, combining honest signed inputs with corrupt inputs received from $\mathcal{A}$ (inserting default values for missing messages), and provides $\boldsymbol{m}^s$ as the sender input to $\mathcal{S}^s_{\mathsf{ua\text{-}bc}}$. Next, $\mathcal{S}$ receives from $\mathcal{S}^s_{\mathsf{ua\text{-}bc}}$ either `continue` or `abort`.
  - If P_1 is corrupted, $\mathcal{S}$ receives a vector $\boldsymbol{m}^s = (m_1^s, \ldots, m_n^s)$ from $\mathcal{S}^s_{\mathsf{ua\text{-}bc}}$, followed by either `continue` or `abort`.

  In addition, $\mathcal{S}$ simulates the honest parties towards $\mathcal{A}$ according to $\mathcal{S}^s_{\mathsf{ua\text{-}bc}}$.
- **Simulating output delivery:** If $\mathcal{S}^s_{\mathsf{ua\text{-}bc}}$ sent `abort`, then $\mathcal{S}$ sends `abort` to the trusted party, and outputs whatever $\mathcal{S}^s_{\mathsf{ua\text{-}bc}}$ outputs. In subsequent rounds, $\mathcal{S}$ simulates honest parties immediately aborting.

  If $\mathcal{S}^s_{\mathsf{ua\text{-}bc}}$ sent `continue`, then $\mathcal{S}$ parses $\boldsymbol{m}^s$ as an n-ary vector and applies the verification rule as the compiler. If some entry fails to verify, then $\mathcal{S}$ sends `abort` to the trusted party, and outputs whatever $\mathcal{A}$ outputs. In subsequent rounds, $\mathcal{S}$ simulates honest parties immediately aborting.

  If all entries pass verification, then $\mathcal{S}$ sends the input values $\{x_j\}_{j \in \mathcal{I}}$ to the functionality followed by `continue`, and outputs whatever $\mathcal{S}^s_{\mathsf{ua\text{-}bc}}$ outputs.

Indistinguishability proof. Round 1 messages sent to P_1 are generated using true honest keys and inputs, making them identically distributed to the real execution. By the security of $\pi_{\mathsf{ua\text{-}bc}}$, the protocol transcript generated by $\mathcal{S}^s_{\mathsf{ua\text{-}bc}}$ is computationally indistinguishable from a real execution transcript of $\pi_{\mathsf{ua\text{-}bc}}$. The output verification function applied by $\mathcal{S}$ is identical to the deterministic rule executed by honest parties in the real protocol, ensuring identical abort decisions. In non-aborting sessions, EUF-CMA security guarantees that no entry corresponding to an honest P_i with $i \notin \mathcal{I}$ contains a value $\tilde{x}_i^s \neq x_i^s$, except with negligible forgery probability. Consequently, the vector delivered in the simulated execution coincides with the vector accepted by honest parties in the real execution.

Replay attacks across distinct sessions are prevented by incorporating the unique session identifier s into every signature payload. Furthermore, the round expansion convention guarantees that

in each session, even if honest parties begin with maximal termination gap, every message transmitted in logical round r is delivered strictly within the listening window for round r, maintaining logical synchronization. Therefore, the one-session indistinguishability argument extends to the full reactive execution by a standard hybrid argument.

Round complexity. Consider the round complexity of the s^{th} session. The compiler uses one initial point-to-point protocol round, followed by one invocation of $\pi_{\mathsf{ua\text{-}bc}}$. This additional protocol round is also expanded according to the same convention. The invocation of $\pi_{\mathsf{ua\text{-}bc}}$ has expected R protocol rounds, so the s^{th} session has expected $R+1$ protocol rounds in total. Since each protocol round of session s is expanded into $2^s - 1$ communication rounds, the expected physical running time of the s^{th} call is $(2^s - 1)(R + 1)$, as claimed. □

Having established a compiler from single-sender UA-broadcast to reactive UA-parallel-broadcast, we can instantiate it with existing protocols. Assuming the existence VRFs (that imply the existence of EUF-CMA signatures), plugging the broadcast protocol of Wan *et al.* [WXSD25] into our compiler yields the following corollary.

Corollary 3.2. *Assuming the existence of VRFs, there exists a reactive UA-parallel-broadcast protocol in the PKI model for the signatures and VRFs that achieves an expected $O(1)$ round complexity per session for $O(1)$ sequential invocations, tolerating any constant corruption threshold $t = (1 - \epsilon)n$ for $0 < \epsilon < 1$.*

4 Expected-Constant-Round UA-MPC Over P2P Channels

In this section, we show how to transform constant-round MPC protocols designed for the broadcast model into protocols operating exclusively over point-to-point channels. The transformed protocol retains expected constant round complexity and security with unanimous abort under any constant fraction of corruptions. Our compiler relies on the reactive parallel-broadcast protocol with unanimous abort established in Section 3.

At a high level, the compiler replaces every broadcast round of the underlying MPC protocol with an invocation of a reactive UA-parallel-broadcast protocol. For clarity, we present the transformation assuming the base MPC protocol communicates solely via broadcast. This assumption is without loss of generality, as point-to-point messaging can be securely instantiated via public-key encryption.

We represent protocols in the broadcast model using next-message functions. At each round, party P_i evaluates $(m_i, y_i) := \text{nxt-msg}(i, x_i, r_i, h_i)$ to compute either its broadcast message m_i or its final output y_i, based on its input x_i, randomness r_i (which includes the setup information), and the history h_i up to that point by evaluating.

Compiler $\mathsf{Comp}_{\mathsf{ua\text{-}mpc}}(\pi_{\mathsf{mpc\text{-}bc}}, \pi_{\mathsf{ua\text{-}rpbc}})$

Private input: Each party P_i holds a private input $x_i \in \{0,1\}^*$.

Setup phase: The setup parameters for $\pi_{\mathsf{mpc\text{-}bc}}$ and $\pi_{\mathsf{ua\text{-}rpbc}}$ are generated by running their respective setup phases. Each party receives its setup information at the onset of the execution. Let r_i denote the setup information for P_i in $\pi_{\mathsf{mpc\text{-}bc}}$ (which includes its random coins).

Notation: Denote by nxt-msg the next-message function of $\pi_{\mathsf{mpc\text{-}bc}}$. Each party P_i maintains a local transcript history $h_i \in \{0,1\}^*$, initialized to the empty string $h_i := \epsilon$.

Protocol: In every round, execute the following steps:

- Party P_i computes $(m_i, y_i) := \text{nxt-msg}(i, x_i, r_i, h_i)$.
  - If $y_i = \epsilon$ (i.e., the output is not ready), call $\pi_{\text{ua-rpbc}}$ with inputs m_i.
    If $\pi_{\text{ua-rpbc}}$ outputs $\perp$, output $\perp$ and halt. Otherwise, let $(\tilde{m}_1, \ldots, \tilde{m}_n)$ denote the output vector, and update $h_i := h_i \circ (\tilde{m}_1, \ldots, \tilde{m}_n)$.
  - If $y_i \neq \epsilon$, output y_i and halt.

Figure 2: Compiler from UA-MPC over broadcast to UA-MPC over P2P.

Theorem 4.1. *Let f be an n-ary function, let $R \in O(\log n)$, and let $\pi_{\text{mpc-bc}}$ be an R-round protocol that t-securely computes f with UA in the broadcast model with straight-line simulation. Let $\pi_{\text{ua-rpbc}}$ be a t-secure reactive UA-parallel-broadcast protocol whose first R sequential calls require expected round complexity E_R.*

Then, $\pi_{\text{mpc}} := \mathsf{Comp}_{\text{ua-mpc}}(\pi_{\text{mpc-bc}}, \pi_{\text{ua-rpbc}})$ t-securely computes f over point-to-point channels with expected round complexity E_R.

Proof. The proof proceeds in two steps. First, we replace the ideal broadcast channel (modeled as RPBC functionality with guaranteed output delivery) in $\pi_{\text{mpc-bc}}$ with an ideal UA-RPBC, showing that the resulting protocol $\pi'_{\text{mpc-bc}}$ remains t-secure. Second, we apply Canetti's composition theorem [Can00] to replace the UA-RPBC functionality with $\pi_{\text{ua-rpbc}}$, yielding the compiled protocol π_{mpc} over point-to-point channels.

We now show that for any PPT t-adversary $\mathcal{A}$ interacting with $\pi'_{\text{mpc-bc}}$, there exists a simulator $\mathcal{S}$ in the ideal UA-MPC execution such that the real and ideal worlds are computationally indistinguishable. Without loss of generality, we assume $\mathcal{A}$ outputs its full execution view (its random tape alongside the ordered sequence of received messages and leakage).

Simulator. Given a PPT adversary $\mathcal{A}$, let $\mathcal{S}_{\text{mpc-bc}}$ denote the simulator guaranteed by the security of $\pi_{\text{mpc-bc}}$. The simulator $\mathcal{S}$ uses $\mathcal{S}_{\text{mpc-bc}}$ to simulate the output of $\mathcal{A}$ in an execution with $\pi_{\text{mpc-bc}}$ and to extract the corrupted parties' inputs.

Because $\mathcal{S}_{\text{mpc-bc}}$ outputs the view of $\mathcal{A}$ in an execution, $\mathcal{S}$ can internally replay $\mathcal{A}$. During replay, if $\mathcal{A}$ sends `abort` to the UA-RPBC functionality, $\mathcal{S}$ forwards `abort` to the ideal UA-MPC functionality and outputs the view prefix up to the abort point. If no broadcast abort occurs, $\mathcal{S}$ outputs the complete view generated by $\mathcal{S}_{\text{mpc-bc}}$.

Indistinguishability proof. Because $\mathcal{A}$ is a valid adversary for $\pi_{\text{mpc-bc}}$, the simulator $\mathcal{S}_{\text{mpc-bc}}$ is well defined. By the security of $\pi_{\text{mpc-bc}}$, the view produced by $\mathcal{S}_{\text{mpc-bc}}$ is computationally indistinguishable from $\mathcal{A}$'s view in a real execution of $\pi_{\text{mpc-bc}}$, together with the corresponding honest parties' outputs in the ideal UA-MPC execution. Moreover, by the assumed straight-line property of $\mathcal{S}_{\text{mpc-bc}}$, this simulated view is a single forward view of $\mathcal{A}$; i.e., it contains a random tape and an ordered sequence of messages and leakage that are consistent with one execution of $\mathcal{A}$.

Up to the first broadcast abort, the ideal GD-RPBC and the ideal UA-RPBC behave identically. Therefore, up to that point, the internal replay matches the transcript produced by $\mathcal{S}_{\text{mpc-bc}}$.

- If no broadcast abort occurs during the internal replay, the execution of $\pi'_{\text{mpc-bc}}$ is identical to the corresponding execution of $\pi_{\text{mpc-bc}}$, and $\mathcal{S}$ outputs the full simulated view generated by $\mathcal{S}_{\text{mpc-bc}}$. Indistinguishability follows directly from the security of $\pi_{\text{mpc-bc}}$.
- If a broadcast abort occurs, let this be the earliest abort round. In a real execution of $\pi'_{\text{mpc-bc}}$, all honest parties output $\perp$ at this point by the guarantees of UA-RPBC, and the adversary's full view is the prefix of its view up to this abort. In the ideal execution, $\mathcal{S}$ sends abort to the ideal UA-MPC functionality at the same point and outputs the corresponding prefix of the simulated view generated by $\mathcal{S}_{\text{mpc-bc}}$.

Since $\mathcal{S}_{\mathsf{mpc\text{-}bc}}$ is straight-line, the distribution of this first-abort point and of the corresponding prefix is the same as in the real execution of $\pi'_{\mathsf{mpc\text{-}bc}}$, up to the distinguishing advantage against $\pi_{\mathsf{mpc\text{-}bc}}$. Thus, both the abort behavior and the adversary's full view are matched.

Therefore, the real execution of $\pi'_{\mathsf{mpc\text{-}bc}}$ with $\mathcal{A}$ is computationally indistinguishable from the ideal UA-MPC execution with $\mathcal{S}$.

Round complexity. If the underlying MPC protocol $\pi_{\mathsf{mpc\text{-}bc}}$ uses $R \in O(\log n)$ broadcast rounds, the worst-case execution (where no premature abort occurs) invokes $\pi_{\mathsf{ua\text{-}rpbc}}$ across R sequential sessions. The expected physical round complexity is therefore exactly E_R point-to-point rounds. □

Instantiating our compiler with an existing constant-round UA-MPC protocol in the broadcast model (e.g., two-round OT-based MPC [GS18, BL18]) alongside the expected-constant-round reactive UA-parallel-broadcast protocol from Corollary 3.2 yields the following corollary.

Corollary 4.2. *Assuming OT and VRFs exist, there exists a UA-MPC protocol in the PKI model for signatures and VRFs that achieves expected-constant round complexity over point-to-point channels for any constant fraction of corruptions.*

5 Round-Complexity Lower Bounds for Broadcast with Abort

In this section, we present our lower bounds. In section 5.1, we extend the classical impossibility results for randomized Byzantine agreement [CMS89, KY84] to broadcast with unanimous abort. In section 5.2, we extend the impossibility result of [GKKO07] to broadcast with identifiable abort.

Our lower bounds apply even to randomized protocols that rely on a private setup. To obtain the strongest possible guarantees, we state these bounds with respect to property-based definitions of broadcast with abort (which we present below), rather than simulation-based notions used earlier in the paper. Furthermore, without loss of generality, it suffices to restrict our attention to a binary message space.

5.1 Impossibility of Constant-Round UA-Broadcast

We begin by recalling the property-based definition of UA-broadcast.

Definition 5.1 (UA-broadcast, property-based definition)**.** *An n-party protocol π, where a designated sender holds an initial input $m \in \{0,1\}$, is a protocol for* broadcast with unanimous abort *if for every static PPT adversary $\mathcal{A}$ corrupting at most t parties, the following conditions hold except for negligible probability:*

- ***Termination.*** *There exists an a-priori-known round bound R such that every honest party terminates within R rounds.*
- ***Consistency.*** *All honest parties output the same value $y \in \{0,1\} \cup \{\perp\}$.*
- ***UA-Validity.*** *If all parties are honest, then every party outputs m. If the sender is honest, then either every honest party outputs m or every honest party outputs $\perp$.*

We proceed to establish strict round-complexity lower bounds for UA-broadcast. The classical indistinguishability arguments in [CMS89, KY84] construct a chain of executions wherein adjacent scenarios are computationally indistinguishable to at least one honest party, while the two extreme endpoints force different sender outputs. We observe that this technique extends directly to protocols with unanimous abort: in the execution where all parties are honest, which is the case in the two endpoints, validity forbids outputting $\perp$, and by the arguments in [CMS89, KY84] the output in both endpoints is the same with high probability. Combined, this yields a contradiction.

Theorem 5.2. *Let $t \in \omega(1)$. Then there is no n-party t-secure UA-broadcast protocol with constant round complexity.*

Proof (sketch). The argument builds on the classical lower-bound framework of Dolev and Strong for deterministic, error-free broadcast [DS83], later generalized to randomized protocols with error in [CMS89, KY84]. This proof forms a sequence of scenarios that each run for $r + 1$ rounds with the following properties:

- In the first scenario, all parties are honest, and the sender holds input 0.
- In any pair of adjacent scenarios, at least one honest party observes an identical view.
- In the final scenario, all parties are honest, and the sender holds input 1.

The total number of scenarios in this sequence is determined by r, t, and n. The construction is recursive: starting from an all-honest execution in which all output 0, Byzantine behavior is introduced round by round; the proof constructs a large (exponential in r) family of neighboring scenarios by recursively modifying the outgoing messages of individual parties while holding at least one honest party's view invariant, the construction forces subsequent executions to maintain output 0. Intuitively, this strategy delays the noticeable effects of Byzantine behavior into later rounds until an honest party's view becomes independent of the sender's input. For the sake of succinctness, we do not get into the details of constructing this sequence of scenarios further since it is the same as in [CMS89, KY84].

In an error-free setting, validity dictates that the first scenario must output 0 and the final scenario must output 1. When allowing for unanimous abort, validity still restricts the output of the first scenario (where all parties are honest and the sender has input 0) strictly to 0. Similarly, the final scenario (where all parties are honest and the sender has input 1) must strictly output 1.

When correctness hold with high probability, the error may accumulate across the scenarios. Chor, Merritt, and Shmoys [CMS89] showed that for any $(r + 1)$-round protocol tolerating t corruptions with $1 \leq r < t$, the maximum error probability ε across the scenario chain satisfies

$$\varepsilon \geq \frac{1}{2}\left(\frac{t}{2nr}\right)^r.$$

For $r \in O(1)$, this lower bound on the error probability cannot be negligible unless t/n is negligible. However, as $n \in \mathsf{poly}(\kappa)$, the ratio t/n is non-negligible. Thus, the error bound can only be avoided if $r \geq t$. This rules out constant-round protocols for $t \in \omega(1)$. □

5.2 Impossibility of Expected-Constant-Round IA-Broadcast

Before presenting our second impossibility result, we present the property-based definition of broadcast with identifiable abort. The definition is the same as Definition 5.1, except that the UA-validity property is replaced by the following:

- **IA-Validity.** If all parties are honest then every party outputs m. If the sender is honest, then either every honest party outputs m or every honest party outputs $(\perp, j)$, where j is an identity of a corrupt party.

We begin by establishing a round-complexity lower bound for the extreme case of $(n - 2)$ corruptions. The proof adapts the chain argument of Garay *et al.* [GKKO07] to account for potential aborts. Specifically, we demonstrate that aborts cannot occur with non-negligible probability in the final scenario; otherwise, the identity of the identified party would propagate backwards through indistinguishable adjacent scenarios, ultimately forcing an honest party to identify itself as cheating in one of the scenarios.

Lemma 5.3. *Let $n \geq 2$. Every n-party $(n-2)$-secure IA-broadcast protocol requires at least $n-1$ rounds with overwhelming probability.*

Proof. Let $P_1, \ldots, P_n$ denote the n parties, where P_1 is the designated sender. Following [GKKO07], for each $1 \leq j \leq n$, define a modified adversarial strategy $\bar{\pi}_j$ derived from π, in which party P_j communicates exclusively with its immediate neighbors along the path $P_1, P_2, \ldots, P_n$. Specifically, in $\bar{\pi}_1$, party P_1 only communicates with P_2; in $\bar{\pi}_n$, party P_n only communicates with P_{n-1}; and for $1 < j < n$, in every intermediate $\bar{\pi}_j$, party P_j only communicates with P_{j-1} and P_{j+1}.

Next, for every $1 \leq i \leq n-1$ and every $b \in \{0,1\}$, we construct scenario S_i^b as follows:

- P_1 begins with input b.
- The honest parties are P_i and P_{i+1} (all other parties are corrupted).
- The honest parties run protocol π, whereas every corrupted party P_j operates based on the strategy $\bar{\pi}_j$.

The scenarios are illustrated in Figure 3.

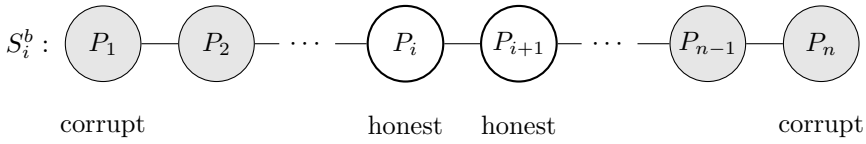


Figure 3: Scenario S_i^b. Only P_i and P_{i+1} execute π honestly; all other parties execute the adversarial strategy $\bar{\pi}_j$.

Notice that for every $2 \leq i \leq n-1$, party P_i is honest in both S_{i-1}^b and S_i^b, and its view is identically distributed across both scenarios.

We first claim that honest parties cannot abort in scenario S_{n-1}^b, except with negligible probability. Suppose toward a contradiction that honest parties abort with non-negligible probability in S_{n-1}^b. By the consistency property of IA-broadcast, honest parties P_{n-1} and P_n must identify the same corrupted party P_{i^*} with non-negligible probability in S_{n-1}^b. Because P_{n-1} has an identical local view in S_{n-2}^b and S_{n-1}^b, party P_{n-1} must also output $(\perp, i^*)$ with non-negligible probability in S_{n-2}^b. By agreement among honest parties in S_{n-2}^b, party P_{n-2} must output $(\perp, i^*)$ as well. Inductively applying this indistinguishability step backward along the chain of scenarios, we deduce that honest parties output $(\perp, i^*)$ with non-negligible probability in every scenario $S_1^b, \ldots, S_{n-1}^b$. This argument is illustrated in Figure 4.

However, across the set of scenarios $\{S_i^b\}_{i=1}^{n-1}$, every party P_j (for $1 \leq j \leq n$) is honest in at least one scenario. IA-validity dictates that honest parties can only output $(\perp, i^*)$ if P_{i^*} is corrupt. Hence, no fixed index i^* can be identified as corrupt across all scenarios, establishing a contradiction. Therefore, honest parties in S_{n-1}^b output $\perp$ with at most negligible probability.

Having proved that none of the above scenarios can abort, the remainder of the proof follows from [GKKO07]. Specifically, we prove that honest parties in S_{n-1}^b must output b except with negligible probability. In scenario S_1^b, the sender P_1 is honest, and therefore IA-validity implies that since the honest parties P_1 and P_2 must output b except with negligible probability (as they do not abort). Since P_2 cannot distinguish S_1^b from S_2^b, party P_2 must also output b in S_2^b, except with negligible probability; by consistency, P_3 must output b in S_2^b as well. Continuing inductively, the honest parties in S_{n-1}^b must output b except with negligible probability.

Finally, consider scenarios S_{n-1}^0 and S_{n-1}^1. Prior to round $n-1$, the view of P_n is independent of the sender's input b, since information from P_1 can traverse the path to P_n only one hop per round. Thus, if P_n terminated before round $n-1$, it could not output both 0 in S_{n-1}^0 and 1 in S_{n-1}^1

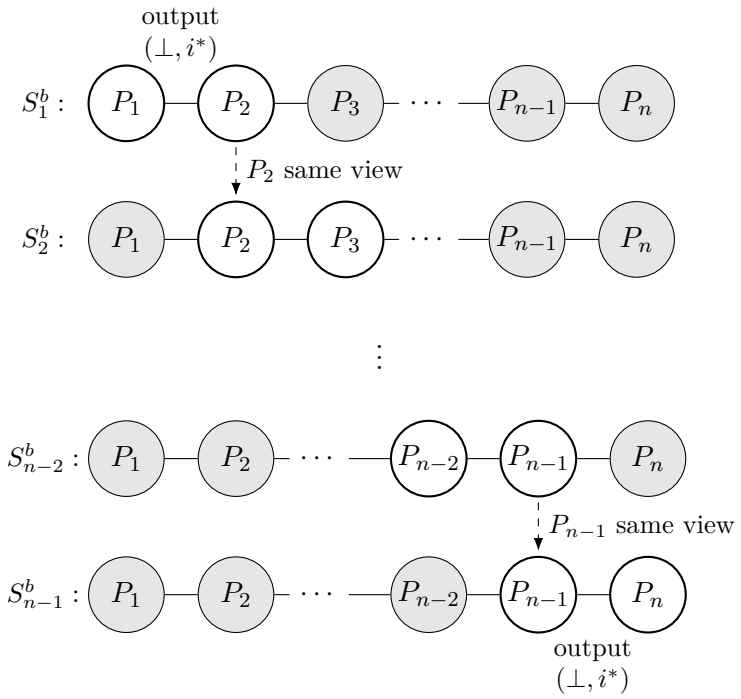


Figure 4: The abort propagation argument. If the honest parties abort and identify the same identity i^* in one scenario, then the same identity propagates through all scenarios, yielding a contradiction.

with overwhelming probability. This contradicts the conclusion above. Hence P_n cannot terminate in fewer than $n-1$ rounds, except with negligible probability. Note that this proof holds for any sender input $b \in \{0,1\}$. See Figure 5 for an illustration of the final step. □

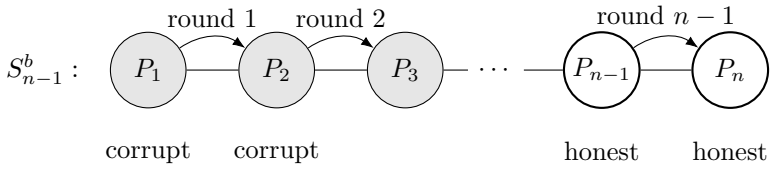


Figure 5: Information propagation in the final scenario S_{n-1}^b.

We now extend the two-honest-party bound of Lemma 5.3 to an arbitrary corruption threshold t satisfying $n-t \geq 2$. Following the player-partitioning technique of [GKKO07]. Instead of arranging individual parties on a path, we arrange sets of parties on a path. Each scenario leaves two adjacent sets honest and corrupts the remaining sets. Denote by $h = n - t$ the number of honest parties. Each set has size at most $h/2$, and the adversary corrupts at most $t = n-h$ parties in each scenario.

Theorem 5.4. *Let $n-t \geq 2$. Every n-party t-secure IA-broadcast protocol must run at least $\frac{2n}{n-t}-1$ rounds with overwhelming probability.*

Proof. Let $h = n - t$ denote the number of honest parties. We reduce the claim to Lemma 5.3 via a standard player-partitioning argument.

Partition the n parties into $N = \frac{2n}{h}$ disjoint sets $G_1, \ldots, G_N$, each of size $h/2$. We construct an N-party protocol Π, where the i^{th} party of Π emulates all parties in G_i running π. Messages sent between sets are delivered between the corresponding emulating parties, while messages internal to a set are simulated locally. The sender of Π emulates the set containing the sender of π.

The outputs of π are mapped to outputs of Π as follows. Every party P_i adopts the output of the party with the minimum index in G_i in the emulated execution: if this output is a bit $v \in \{0, 1\}$, then P_i outputs v; if this output is $(\perp, i^*)$, then P_i outputs $(\perp, \gamma(i^*))$, where $\gamma(i^*) \in [N]$ is such that $P_{i^*} \in G_{\gamma(i^*)}$.

We claim that Π is an N-party, $(N-2)$-secure IA-broadcast protocol. Any adversary corrupting at most $N - 2$ parties in Π corresponds to an adversary for π that corrupts all parties in the corresponding sets. Since at least two sets remain honest, and each set has size $h/2$, the number of honest parties in the induced execution of π is at least h, which means the adversary corrupts at most $n - h = t$ parties in π. Thus, the security of π against t-adversaries implies termination and consistency for Π. Validity for non-abort outputs is inherited directly from the validity of π. In case of abort, if an honest party of Π outputs $(\perp, \ell)$, then by construction the corresponding emulated execution of π outputs $(\perp, i^*)$ for some $i^* \in G_\ell$. Since π satisfies IA-validity, party P_{i^*} is corrupt in the induced execution of π. Hence G_ℓ is a corrupted set, and therefore the ℓ^{th} party of Π is corrupt. Thus Π satisfies IA-validity.

By Lemma 5.3, every N-party, $(N - 2)$-secure IA-broadcast protocol must run for at least $N - 1$ rounds with overwhelming probability. Applying the lemma to Π, we conclude that Π, and therefore also π, must run for at least $N - 1$ rounds with overwhelming probability. Substituting $N = \frac{2n}{h}$ and $h = n - t$ implies that π must run for at least $\frac{2n}{n-t} - 1$ rounds with overwhelming probability. □

Bibliography

[ACGJ18] Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Round-optimal secure multiparty computation with honest majority. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 395–424. Springer, Cham, August 2018.

[ACGJ19] Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Two round information-theoretic MPC with malicious security. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 532–561. Springer, Cham, May 2019.

[BE03] Michael Ben-Or and Ran El-Yaniv. Resilient-optimal interactive consistency in constant time. *Distributed Computing*, 16(4):249–262, 2003.

[Ben83] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In Robert L. Probert, Nancy A. Lynch, and Nicola Santoro, editors, *2nd ACM PODC*, pages 27–30. ACM, August 1983.

[BGP89] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Towards optimal distributed consensus (extended abstract). In *30th FOCS*, pages 410–415. IEEE Computer Society Press, October / November 1989.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[BL18] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 500–532. Springer, Cham, April / May 2018.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

[Bra84] Gabriel Bracha. An asynchronous [(n-1)/3]-resilient consensus protocol. In Robert L. Probert, Nancy A. Lynch, and Nicola Santoro, editors, *3rd ACM PODC*, pages 154–162. ACM, August 1984.

[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 *20th ACM STOC*, pages 11–19. ACM Press, May 1988.

[CCGZ19] Ran Cohen, Sandro Coretti, Juan A. Garay, and Vassilis Zikas. Probabilistic termination and composability of cryptographic protocols. *Journal of Cryptology*, 32(3):690–741, July 2019.

[CCGZ21] Ran Cohen, Sandro Coretti, Juan A. Garay, and Vassilis Zikas. Round-preserving parallel composition of probabilistic-termination cryptographic protocols. *Journal of Cryptology*, 34(2):12:1–12:57, March 2021.

[CDL+25] Ran Cohen, Jack Doerner, Eysa Lee, Anna Lysyanskaya, and Lawrence Roy. An unstoppable ideal functionality for signatures and a modular analysis of the dolev-strong broadcast. In Benny Applebaum and Huijia (Rachel) Lin, editors, *TCC 2025, Part IV*, volume 16271 of *LNCS*, pages 675–708. Springer, Cham, December 2025.

[CFG+23] Ran Cohen, Pouyan Forghani, Juan A. Garay, Rutvik Patel, and Vassilis Zikas. Concurrent asynchronous byzantine agreement in expected-constant rounds, revisited. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part IV*, volume 14372 of *LNCS*, pages 422–451. Springer, Cham, November / December 2023.

[CGZ20] Ran Cohen, Juan A. Garay, and Vassilis Zikas. Broadcast-optimal two-round MPC. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 828–858. Springer, Cham, May 2020.

[CGZ23] Ran Cohen, Juan A. Garay, and Vassilis Zikas. Completeness theorems for adaptively secure broadcast. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 3–38. Springer, Cham, August 2023.

[CHM+22] Ran Cohen, Iftach Haitner, Nikolaos Makriyannis, Matan Orland, and Alex Samorodnitsky. On the round complexity of randomized byzantine agreement. *Journal of Cryptology*, 35(2):10, April 2022.

[CL17] Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. *Journal of Cryptology*, 30(4):1157–1186, October 2017.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th ACM STOC*, pages 364–369. ACM Press, May 1986.

[CMS89] Benny Chor, Michael Merritt, and David B. Shmoys. Simple constant-time consensus protocols in realistic failure models. *Journal of the ACM*, 36(3):591–614, 1989.

[CPS20] T.-H. Hubert Chan, Rafael Pass, and Elaine Shi. Sublinear-round byzantine agreement under corrupt majority. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 246–265. Springer, Cham, May 2020.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 378–394. Springer, Berlin, Heidelberg, August 2005.

[DLM82] Richard A. DeMillo, Nancy A. Lynch, and Michael Merritt. Cryptographic protocols. In *14th ACM STOC*, pages 383–400. ACM Press, May 1982.

[DN07] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 572–590. Springer, Berlin, Heidelberg, August 2007.

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for Byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[Du89] Dingzhu Du. Lower bounds for weak byzantine agreement. *Acta Mathematicae Applicatae Sinica*, 5:355–360, 1989.

[FG03] Matthias Fitzi and Juan A. Garay. Efficient player-optimal protocols for strong and differential consensus. In Elizabeth Borowsky and Sergio Rajsbaum, editors, *22nd ACM PODC*, pages 211–220. ACM, July 2003.

[Fit03] Matthias Fitzi. *Generalized communication and security models in Byzantine agreement*. PhD thesis, ETH Zurich, Zürich, Switzerland, 2003.

[FL82] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14(4):183–186, 1982.

[FM88] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *20th ACM STOC*, pages 148–161. ACM Press, May 1988.

[FN09] Matthias Fitzi and Jesper Buus Nielsen. On the number of synchronous rounds required for byzantine agreement. In *International Symposium on Distributed Computing (DISC)*, pages 449–463, 2009.

[GKKO07] Juan A. Garay, Jonathan Katz, Chiu-Yuen Koo, and Rafail Ostrovsky. Round complexity of authenticated broadcast with a dishonest majority. In *48th FOCS*, pages 658–668. IEEE Computer Society Press, October 2007.

[GKKZ11] Juan A. Garay, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. Adaptively secure broadcast, revisited. In Cyril Gavoille and Pierre Fraigniaud, editors, *30th ACM PODC*, pages 179–186. ACM, June 2011.

[GL05] Shafi Goldwasser and Yehuda Lindell. Secure multi-party computation without agreement. *Journal of Cryptology*, 18(3):247–287, July 2005.

[GM93] Juan A. Garay and Yoram Moses. Fully polynomial byzantine agreement in t+1 rounds. In *25th ACM STOC*, pages 31–41. ACM Press, May 1993.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[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 Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[GS18] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 468–499. Springer, Cham, April / May 2018.

[HZ10] Martin Hirt and Vassilis Zikas. Adaptively secure broadcast. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 466–485. Springer, Berlin, Heidelberg, May / June 2010.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 369–386. Springer, Berlin, Heidelberg, August 2014.

[KK06] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 445–462. Springer, Berlin, Heidelberg, August 2006.

[KY84] Anna R. Karlin and Andrew Chi-Chih Yao. Probabilistic lower bounds for Byzantine agreement and clock synchronization. Unpublished manuscript, 1984.

[LLR02] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. Sequential composition of protocols without simultaneous termination. In Aleta Ricciardi, editor, *21st ACM PODC*, pages 203–212. ACM, July 2002.

[LLR06] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. On the composition of authenticated byzantine agreement. *Journal of the ACM*, 53(6):881–917, 2006.

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th FOCS*, pages 120–130. IEEE Computer Society Press, October 1999.

[PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

[Rab83] Michael O. Rabin. Randomized byzantine generals. In *24th FOCS*, pages 403–409. IEEE Computer Society Press, November 1983.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st ACM STOC*, pages 73–85. ACM Press, May 1989.

[WXSD25] Jun Wan, Hanshen Xiao, Elaine Shi, and Srinivas Devadas. Expected constant round byzantine broadcast under dishonest majority. *Journal of the ACM*, 72(4):30:1–30:39, 2025.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.