Information-Theoretic Network-Agnostic MPC with Polynomial Communication

Xiaoyu Ji[1], Chen-Da Liu-Zhang[2], Daniel Pöllmann[3], and Yifan Song[4]

[1] jixy23@mails.tsinghua.edu.cn, Tsinghua University
[2] chen-da.liuzhang@hslu.ch, Lucerne University of Applied Sciences and Arts
[3] daniel.poellmann@hslu.ch, Lucerne University of Applied Sciences and Arts & ETH Zurich
[4] yfsong@mail.tsinghua.edu.cn, Tsinghua University and Shanghai Qi Zhi Institute

Abstract. Network-agnostic MPC protocols tolerate simultaneously a higher number of corruptions $t_s < n/2$ when the network is synchronous, and a lower number $t_a < n/3$ when the network is asynchronous. As such, they provide strong resilience, irrespective of the type of underlying communication network.

We focus on improving the communication complexity of network-agnostic MPC with optimal resilience $2t_s + t_a < n$. In this regime, there are no polynomial-time information-theoretic solutions and current computational protocols (without fully-homomorphic encryption) communicate $O(n^2)$ elements per multiplication gate.

In this work, we significantly advance the landscape by introducing the first information-theoretic protocol with quadratic communication per multiplication gate and the first computational protocol with linear communication per multiplication gate based solely on signatures and symmetric-key encryption.

1 Introduction

The goal of Multi-Party Computation (MPC) is to enable a set of n parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ to jointly compute any arbitrary function on private inputs [Yao82, BGW88, GMW87, CCD88]. The computation is performed in such a way that an adversary does not learn anything beyond the output.

Protocols for MPC can be roughly classified into two models, depending upon the network they operate in. In the synchronous network model, the parties proceed synchronously in rounds. In each round, parties send messages and receive the messages by the end of the round. To implement this round structure, parties are assumed to have synchronized clocks, so they can all start rounds at the same time, and it is also assumed that messages are delivered within a time upper bound Δ, which is an a priori publicly known parameter. In this model, it is possible to achieve MPC for up to $t_s < \frac{n}{2}$ corrupted parties with guaranteed output delivery [RB89, CDD+99, CDN01, DN03, DI05] assuming correlated randomness or broadcast channels. However, the downside is that the protocol may lose all security guarantees if a message is not delivered in time, making this model problematic in the real world, where networks may occasionally add significant unpredictable delays.

In the asynchronous network model, protocols do not rely on any delay upper-bound, which makes them suitable for even the most unpredictable network conditions. However, this comes at the cost of resilience, and protocols in the asynchronous network model can only support up to $t_a < \frac{n}{3}$ corrupted parties [BKR94, PCR09, CP15, CGHZ16, HNP05, Coh16, CHLZ21, GLZS24, BJK+25b, BJK+25a]

This gave rise to the network-agnostic MPC line of work [BZL20, DHLZ21, BCLZL23, ACC22, AC23, BCV24, PP24] which achieves security guarantees in either network scenario. More precisely, a network-agnostic MPC simultaneously tolerates up to t_s corruptions when the network is synchronous, and t_a corruptions when the network is asynchronous.

Communication Complexity of Network-Agnostic MPC. In the computational setting assuming threshold homomorphic encryption tools and non-interactive zero-knowledge, the first network-agnostic protocols [BZL20, DHLZ21] achieved a communication complexity of $O(n^3)$ field elements per multiplication gate with optimal resilience $t_a + 2t_s < n$. Under the same computational assumptions, the work [BCLZL23]

achieved $O(n^2)$ elements per gate. The recent work [BCV24] only makes use of linearly-homomorphic commitments and a PKI for signatures. However, the communication complexity is $O(n^7)$ elements per multiplication gate.

In the information-theoretic setting with optimal resilience $t_a + 2t_s < n$, there are no network-agnostic MPC protocols [AC23] that run in polynomial time against threshold adversaries. The work [ACC22] introduced a polynomial solution communicating $O(n^5/t_a)$ elements in the perfect security setting and $t_a+3t_s < n$ resilience, and later [PP24] improved the resilience to $2\max(t_s, t_a)+\max(2t_a, t_s) < n$ (albeit with exponential communication).

Compared to the monolithic network setting, recent works in the purely asynchronous setting achieved linear communication per multiplication gate [AAPP24, GLZS24] even with information-theoretic security. This leads to the question of how low one can achieve a similar result with network-agnostic security, or whether there is an inherent communication overhead needed.

Is information-theoretic network-agnostic MPC with linear communication and optimal resilience possible?

1.1 Contributions

We significantly advance the state of the art in the communication complexity of network-agnostic MPC by providing several protocols with optimal resilience. Note that in the setting of information-theoretic protocols with optimal resilience, there are no known solutions even with polynomial time.

Our first result is an information-theoretic MPC with polynomial time and communication as well as optimal resilience. More concretely, our protocol incurs a quadratic communication term next to the circuit size (albeit larger dependencies in the depth). The protocol makes use of a trusted setup for pseudo signatures.

Theorem 1. *Let n, t_s, t_a be positive integers such that $n = 2t_s + t_a + 1$ and κ be the security parameter. Let f be an n-ary function. Assuming a trusted setup for pseudo signatures, there is an unconditionally secure network-agnostic MPC protocol for f among n parties against a malicious adversary who controls up to t_s corrupted parties when the network is synchronous, and t_a corrupted parties when the network is asynchronous.*

For a function f with circuit size $|C|$, input size C_I, output size C_O, and circuit depth D, the MPC protocol requires setup size $O(t_a(n|C|\kappa + n^3C_I\kappa + n^3C_O\kappa) + n^6D\kappa + n^9\kappa)$ and achieves communication complexity $O(n|C|\kappa + n^3C_I\kappa + n^3C_O\kappa + (n^3 + D)n^6\kappa\log(nD) + n^8\kappa^2)$ in expectation in the optimistic case and $O(t_a(n|C|\kappa + n^3C_I\kappa + n^3C_O\kappa) + n^8\kappa^2 \cdot D + n^{14}\kappa^2)$ in expectation in the worst case.

We also introduce a computational MPC protocol with linear communication (in the circuit size), but still with larger dependencies in the depth. The protocol also achieves optimal resilience and incurs minimal computational assumptions (only signatures and symmetric-key encryption). The protocol assumes a trusted setup for PKI[5].

Theorem 2. *Let n, t_s, t_a be positive integers such that $n = 2t_s + t_a + 1$ and κ be the security parameter. Let f be an n-ary function. Assuming PKI and symmetric-key encryption schemes, there is a computationally secure network-agnostic MPC protocol for f among n parties against a malicious adversary who controls up to t_s corrupted parties when the network is synchronous, and t_a corrupted parties when the network is asynchronous.*

For a function f with circuit size $|C|$, input size C_I, output size C_O, and circuit depth D, the MPC protocol requires setup size $O(n^2\kappa)$ and achieves communication complexity $O(n|C|\kappa+n^3C_I\kappa+n^3C_O\kappa+(n^3+D)n^6\kappa\log(nD)+n^8\kappa^2)$ in expectation in the optimistic case and $O(n|C|\kappa+n^3C_I\kappa+n^3C_O\kappa+n^7\kappa^2\cdot D+n^{14}\kappa^2)$ in expectation in the worst case.

[5] Our construction only uses a signature scheme with signature size linearly in the message size.

1.2 Related Work

The network agnostic model was introduced by Blum, Katz and Loss [BKL19] in the context of consensus protocols. After this, several network-agnostic protocols have been proposed for different types of primitives, including state-machine replication [BKL19, AMN+20, BKL21, MR21], consensus and broadcast protocols [DHLZ21, BKL21, ABKL22, DME24], approximate and convex agreement [GLZW22, GLZW23, CGWW24], secure-message transmission [BCV23, DLZ23], distributed key generation [BCLZL23] and MPC [BZL20, DHLZ21, BCLZL23, ACC22, AC23, BCV24, PP24].

The first network-agnostic (cryptographic) MPC was introduced by Blum, Liu-Zhang and Loss [BZL20], which achieves the optimal resilience of $t_a + 2t_s < n$, assuming a trusted setup for threshold additive homomorphic encryption and NIZKs. In the same setting, the work [DHLZ21] proposed a round-efficient solution. Both these works incur $O(n^3)$ per multiplication. The work [BCLZL23] introduced a network-agnostic solution for DKG, which was used to bootstrap the trusted setup of (a more efficient version) of the protocol in [BZL20] from plain bulletin-board PKI, albeit with the same computational assumptions. With the bootstrapping, the incurred cost is still $O(n^3)$ per multiplication; but assuming the trusted setup the authors improved the complexity to $O(n^2)$. The recent work [BCV24] reduced the computational assumptions to linear commitments and signatures, but at the cost of increased communication $O(n^7)$ per multiplication. We further note that assuming fully-homomorphic encryption, it is straightforward to achieve communication complexity independent of the circuit size.

The works [ACC22] introduced provide perfectly secure network-agnostic MPC assuming $t_a + 3t_s < n$ communicating $O(n^5/t_a)$ per multiplication gate. This result was extended to the general adversary setting with perfect security [ACC23] and statistical security [AC23], although the communication complexity grows with the size of the adversary structure, and therefore is exponential for usual choices of threshold adversaries. The work [PP24] defines a perfectly secure MPC protocol under the weaker condition $2\max(t_s, t_a) + \max(2t_a, t_s) < n$, but the efficiency is also exponential.

2 Technical Overview

Recall that we use n for the number of parties, t_s for the synchronous corruption threshold, and t_a for the asynchronous corruption threshold. We further assume that $t_a \leq t_s$ and $n = 2t_s + t_a + 1$. Let Δ be the delivery time-bound for a message sent in the synchronous network.

2.1 Overview of Our Construction

We start with a synchronous MPC protocol $\Pi_{\texttt{SMPC-GOD}}$ that is secure against t_s corrupted parties with guaranteed output delivery. At a high level, we want to let all parties attempt to execute $\Pi_{\texttt{SMPC-GOD}}$ in the unknown network setting as follows: (1) In each round, all parties follow $\Pi_{\texttt{SMPC-GOD}}$ to send messages to each other. (2) Each party, after receiving all messages from other parties, moves on to the next round.

However, this idea may fail entirely when the network is asynchronous due to the following two problems:

- First, a party may not be able to receive all messages from other parties.

When a party A does not receive the desired message from another party B, A cannot distinguish a corrupted B not sending the message from an honest B whose message is delayed due to the large network latency. In the case that B is corrupted, A may wait forever and we lose liveness.

- Second, even if a party has received all messages from other parties in the current round, there is no guarantee that all other parties also have received the desired messages in this round.

Thus, when a party A moves to the next round, another party B may still wait for messages in the current round. However, the security of $\Pi_{\texttt{SMPC-GOD}}$ may highly rely on the property that when an honest party moves on to the next round, all other honest parties have received their messages in the current round.

Starting Idea. We note that to preserve the security of $\Pi_{\texttt{SMPC-GOD}}$, it is sufficient to let each party broadcast a confirmation (by using a network-agnostic broadcast protocol $\Pi_{\texttt{RBC}}$) claiming that he has received all desired messages in the current round. Then a party only moves to the next round if he has received the confirmations from all parties (including himself). In this way, when an honest party proceeds to the next round, all honest parties have received the desired messages in the current round. Furthermore, by the guarantee of $\Pi_{\texttt{RBC}}$, all honest parties will eventually receive all confirmations and move to the next round. This simple modification allows us to prevent an adversary breaking the security of $\Pi_{\texttt{SMPC-GOD}}$ in the asynchronous setting.

However, it does not resolve the issue of liveness: (1) an honest party A may never receive all desired messages in the current round due to some corrupted party B not sending his message to A, and (2) a corrupted party A may never broadcast the confirmation even if he has received all messages, causing all parties waiting for the confirmations forever. It appears that fully resolving this issue is very difficult or even impossible since in the asynchronous setting, the best resilience one can hope is $t_a < n/3$ whereas in the synchronous setting, the best resilience is $t_s < n/2$.

At the heart of our technique, we propose an ideal functionality $\mathcal{F}_{\mathsf{Channel}}$ that helps all parties pass their messages in each round. Very informally, $\mathcal{F}_{\mathsf{Channel}}$ ensures that for every pair of parties (A, B) where A is the receiver and B is the sender, either (1) A receives his message from B, or (2) all parties identify a party $C \in \{A, B\}$, which is guaranteed to be corrupted if the network is synchronous (but might be an honest party if the network is asynchronous). We refer to this property as *synchronous identifiable abort*.

With $\mathcal{F}_{\mathsf{Channel}}$, in each round, every pair of parties (A, B) use $\mathcal{F}_{\mathsf{Channel}}$ to exchange their messages. Then at the end of this round, all parties would obtain a subset W of identified parties. By the guarantee of $\mathcal{F}_{\mathsf{Channel}}$, for every pair of parties (A, B) that are not in W, they have received desired messages from each other. To deal with parties in W, we note that:

- When the network is synchronous, parties in W are guaranteed to be corrupted. Thus, we may eliminate all parties in W and assume that messages sent by those parties are all-0 strings. This will not break the security of the underlying synchronous protocol $\Pi_{\texttt{SMPC-GOD}}$.
- When the network is asynchronous, however, W may contain honest parties. Recall that when the network is asynchronous, the number of corrupted parties is bounded by t_a. On the other hand, $\Pi_{\texttt{SMPC-GOD}}$ can tolerate up to $t_s \geq t_a$ corrupted parties. Thus, as long as $|W| \leq t_s - t_a$, we can still eliminate parties in W and set their messages to be all 0-strings.
  Effectively, we just view parties in W as corrupted parties. This is equivalent to an adversary of $\Pi_{\texttt{SMPC-GOD}}$ who first corrupts t_a parties, and then *adaptively* corrupts parties in W in this round. When $\Pi_{\texttt{SMPC-GOD}}$ achieves adaptive security, as long as the total number of eliminated parties is bounded by $t_s - t_a$, the security of $\Pi_{\texttt{SMPC-GOD}}$ remains.

Thus, when $|W| \leq t_s - t_a$, all parties eliminate parties in W and take all 0-strings as messages sent by those parties. When $|W| > t_s - t_a$, further removing parties in W may break the security of $\Pi_{\texttt{SMPC-GOD}}$ when the network is asynchronous since the total number of effective corrupted parties would be larger than t_s. However, we notice that after eliminating $t_s - t_a$ corrupted parties:

- In the synchronous setting, there are $n - (t_s - t_a) = t_s + 2t_a + 1$ remaining parties and at most $t_s - (t_s - t_a) = t_a$ parties are corrupted (since all eliminated parties are corrupted). In particular, we have $t_s + 2t_a + 1 \geq 3t_a + 1$. Thus for the remaining parties, we are effectively in the $1/3$ corruption case.
- Similarly, in the asynchronous setting, there are $n - (t_s - t_a) = t_s + 2t_a + 1$ remaining parties and at most t_a parties are corrupted (since all eliminated parties can be honest in the worst case). We are still in the $1/3$ corruption case.

Thus, after eliminating $t_s - t_a$ corrupted parties, all parties can fall back to an asynchronous MPC protocol running among the remaining parties with $1/3$ corruption threshold.

2.2 Realization of $\mathcal{F}_{\mathsf{Channel}}$

To realize $\mathcal{F}_{\mathsf{Channel}}$, we first focus on a P2P message from a sender B to a receiver A. As we have discussed above, we want to ensure that either (1) A receives his message from B, or (2) all parties identify a party

$C \in \{A, B\}$, which is guaranteed to be corrupted if the network is synchronous (but might be an honest party if the network is asynchronous).

We first assume that when the network is synchronous, for each round, all honest parties enter this round at a similar time. To be more concrete, we assume that the time difference between the starting time of any two honest parties is bounded by Δ. In our compiler, we will ensure this property whenever $\mathcal{F}_{\mathsf{Channel}}$ is invoked when the network is synchronous.

Our construction assumes a signature scheme and makes use of a network-agnostic broadcast protocol $\Pi_{\mathtt{RBC}}$ and a network-agnostic agreement protocol $\Pi_{\mathtt{BA}}$ with the following property:

- When the network is synchronous, if an honest party terminates $\Pi_{\mathtt{RBC}}$ (or $\Pi_{\mathtt{BA}}$), then all honest parties terminate $\Pi_{\mathtt{RBC}}$ (or $\Pi_{\mathtt{BA}}$) within Δ time.

We use $\mathtt{Time}_{\mathsf{bc}}$ for the running time of $\Pi_{\mathtt{RBC}}$ when the network is synchronous and the dealer is honest. In Section 4.3 and Section 4.2, we give concrete instantiations of $\Pi_{\mathtt{RBC}}$ and $\Pi_{\mathtt{BA}}$. At the beginning of the execution, we assume a trusted setup for signatures as described in $\mathcal{F}_{\mathsf{Setup}}$.

Functionality $\mathcal{F}_{\mathsf{Setup}}$

$\mathcal{F}_{\mathsf{Setup}}$ runs with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an adversary $\mathcal{S}$. Let $(\mathtt{Gen}, \mathtt{Sign}, \mathtt{Vrfy})$ be a signature scheme.

1: For each $P_i \in \mathcal{P}$, $\mathcal{F}_{\mathsf{Setup}}$ runs $\mathtt{Gen}(1^\kappa)$ to generate a pair of signing key and verification keys $(\mathsf{sk}^{(i)}, \{\mathsf{vk}_j^{(i)}\}_{j=1}^n)$. Then $\mathcal{F}_{\mathsf{Setup}}$ generates a request-based delayed output to send P_i the signing key $\mathsf{sk}^{(i)}$, and generates a request-based delayed output to send each $P_j \in \mathcal{P}$ the verification key $\mathsf{vk}_j^{(i)}$.

Initial Attempt. At the beginning of each round, A tries to receive the message from B directly. Let $\mathtt{rid}$ denote the index of the current round.

Protocol $\Pi_{\mathtt{P2P\text{-}Send}}(A, B, \mathtt{rid})$

Initial Attempt

1: After B enters the $\mathtt{rid}$-th round, B sends his message m to A.
2: After A enters the $\mathtt{rid}$-th round, A waits for 2Δ time for the message m from B. If A receives the message from B within the time limit, A broadcasts a confirmation $(\mathtt{Confirm}, A, B, \mathtt{rid})$.

Here we set the waiting time to be 2Δ since B may start later than A by Δ, and the message sent by B may require another Δ time to be delivered.

Resend Attempt. Now consider the case where A does not receive the message from B within the time limit. There are two possibilities: either B is corrupted or the network is asynchronous. As observed in [BCV24], for the first case, there is no need to protect the secrecy of B's message. For the second case, we still need to protect the secrecy of B's message but only against t_a corrupted parties.

Thus, to move on, A will ask B to resend his message by distributing a degree-t_a Shamir secret sharing of the message to all parties. Then A will try to collect enough shares from all parties to reconstruct the message. In more details, all parties run the following steps.

Protocol $\Pi_{\mathtt{P2P\text{-}Send}}(A, B, \mathtt{rid})$

Resend Attempt

1: The receiver A broadcasts $(\mathtt{Resend}, A, B, \mathtt{rid})$ to all parties.
2: After entering the $\mathtt{rid}$-th round and receiving $(\mathtt{Resend}, A, B, \mathtt{rid})$ broadcast by A, the sender B samples a random degree-t_a Shamir sharing of his message m, denoted by $[m]_{t_a} := (m_1, \ldots, m_n)$. Then B sends $(\mathtt{Resend}, A, B, \mathtt{rid}, m_i)$ together with his signature to each P_i.
3: For each party P_i, after entering the $\mathtt{rid}$-th round and receiving $(\mathtt{Resend}, A, B, \mathtt{rid})$ broadcast by A, if P_i receives (or has received) $(\mathtt{Resend}, A, B, \mathtt{rid}, m_i)$ with a valid signature from B within 2Δ time, P_i sends

$(\texttt{Resend}, A, B, \texttt{rid}, m_i)$ and the signature of B to A. Otherwise, P_i sends $(\texttt{Dispute}, P_i, B)$ together with his signature to A.

4: Recall that $\texttt{Time}_{\texttt{bc}}$ is the time required by $\Pi_{\texttt{RBC}}$ when the dealer is honest and the network is synchronous. The receiver A, after starting $\Pi_{\texttt{RBC}}$ for $(\texttt{Resend}, A, B, \texttt{rid})$, waits for the response from each party P_i for $\texttt{Time}_{\texttt{bc}} + 4\Delta$ time. A initializes a set $\texttt{Proof} = \varnothing$.
- If A receives $(\texttt{Resend}, A, B, \texttt{rid}, m_i)$ signed by B from P_i, A records m_i.
- If A receives $(\texttt{Dispute}, P_i, B)$ with a valid signature from P_i, A adds $(\texttt{Dispute}, P_i, B)$ and the signature of P_i to $\texttt{Proof}$.
- Otherwise, A records $(\texttt{Dispute}, A, P_i)$.

Finally, A invokes an instance of $\Pi_{\texttt{RBC}}$ to broadcast a unique message determined as follows:
- If A records m_i from $t_a + 1$ distinct P_i, A reconstructs the secret m and broadcasts a confirmation $(\texttt{Confirm}, A, B, \texttt{rid})$.
- If $|\texttt{Proof}| \geq t_s + 1$, A broadcasts $(\texttt{Eliminate}, B, \texttt{Proof})$.
- Otherwise, A broadcasts $(\texttt{Eliminate}, A)$.

Note that the last case happens only when A receives at most t_a shares and $|\texttt{Proof}| \leq t_s$, which implies that A recorded $(\texttt{Dispute}, A, P_i)$ for at least $n - t_a - t_s \geq t_s + 1$ distinct P_i.

The key insight of the resending attempt is that a party is eliminated if and only if he has a dispute with at least $t_s + 1$ parties. We ensure that if the network is synchronous, each pair of disputed parties contains at least one corrupted party. Thus, if a party has dispute with $t_s + 1$ parties, this party must be corrupted.

Now we show that if the network is synchronous, no two honest parties will be set dispute in the above process.

- For each pair of honest (P_i, B), when P_i enters the $\texttt{rid}$-th round and receives $(\texttt{Resend}, A, B, \texttt{rid})$ broadcast by A, B is guaranteed to enter the $\texttt{rid}$-th round and receive $(\texttt{Resend}, A, B, \texttt{rid})$ broadcast by A within Δ time. This also ensures that P_i will receive $(\texttt{Resend}, A, B, \texttt{rid}, m_i)$ with a valid signature from B within 2Δ time. Thus, P_i will never send $(\texttt{Dispute}, P_i, B)$ to A.
- For each pair of honest (A, P_i), A will broadcast $(\texttt{Resend}, A, B, \texttt{rid})$ after A enters the $\texttt{rid}$-th round. Then P_i is guaranteed to enter the $\texttt{rid}$-th round and receives $(\texttt{Resend}, A, B, \texttt{rid})$ from A within Δ time after A terminates $\Pi_{\texttt{RBC}}$. By construction, after another 2Δ time, P_i responses to A either $(\texttt{Resend}, A, B, \texttt{rid}, m_i)$ signed by B or $(\texttt{Dispute}, P_i, B)$. In either case, A will get the response from P_i within 4Δ time and thus A will never record $(\texttt{Dispute}, A, P_i)$.

Preserving Liveness. We note that the above process still does not ensure liveness when A is corrupted: A may never broadcast the final message. To preserve liveness, all parties will accuse A if he does not respond in time.

Protocol $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$

Termination

1: Recall that $\texttt{Time}_{\texttt{bc}}$ is the time required by $\Pi_{\texttt{RBC}}$ when the dealer is honest and the network is synchronous. All parties invoke $\Pi_{\texttt{BA}}$ and decide their input as follows: After entering the $\texttt{rid}$-th round, each party P_i waits for $7\Delta + 2\texttt{Time}_{\texttt{bc}}$ for the final broadcast message from A.
- If the received broadcast message is in the form of $(\texttt{Confirm}, A, B, \texttt{rid})$, $(\texttt{Eliminate}, B, \texttt{Proof})$ (with a valid $\texttt{Proof}$), or $(\texttt{Eliminate}, A)$, P_i sets his input of $\Pi_{\texttt{BA}}$ to be 1.
- Otherwise P_i sets his input of $\Pi_{\texttt{BA}}$ to be 0.

2: All parties decide their output based on the result of $\Pi_{\texttt{BA}}$ and the final broadcast message from A.
- If $\Pi_{\texttt{BA}}$ outputs 1 and the final broadcast message from A is $(\texttt{Confirm}, A, B, \texttt{rid})$, all parties output $(\texttt{Confirm}, A, B, \texttt{rid})$ and A outputs the received message m in addition.
- If $\Pi_{\texttt{BA}}$ outputs 1 and the final broadcast message from A is $(\texttt{Eliminate}, B, \texttt{Proof})$ (with a valid $\texttt{Proof}$), all parties output $(\texttt{Eliminate}, B)$.
- Otherwise, all parties output $(\texttt{Eliminate}, A)$.

The analysis of time limit $7\Delta + 2\texttt{Time}_{\texttt{bc}}$ is shown in Section D.1. When the network is synchronous, an honest P_i will never set his input of $\Pi_{\texttt{BA}}$ to be 0 for an honest A. When the network is asynchronous, we want

agreement on the output of honest parties, that is, all honest parties either agree on that A has received the message m from B or they agree on a party $C \in \{A, B\}$ to be eliminated. The BA protocol, to which honest parties input their bit at time $7\Delta + 2\texttt{Time}_{\texttt{bc}}$, guarantees that all honest parties will receive the same output bit and will choose their output accordingly. When the bit is 0, they will all eliminate A, while when the bit is 1, they will accept the output of $\Pi_{\texttt{RBC}}$ as their output. In this case, $\Pi_{\texttt{RBC}}$ is guaranteed to terminate as it already terminated for at least one honest party—otherwise BA must have had output 0 due to the validity of BA. Further, we want honest parties to eventually terminate and output the aforementioned decision. It follows directly that all honest parties will terminate $\Pi_{\texttt{P2P-Send}}$ eventually because all honest parties give input to the BA protocol at time $7\Delta + 2\texttt{Time}_{\texttt{bc}}$, which is therefore guaranteed to terminate.

We note that to send a message m of length $|m|$, the resend attempt requires B to send a share of size $|m|$ to each party, which is then passed to A. This results in an $O(n)$ multiplicative factor in the communication complexity. Ideally, A only needs $t_a + 1$ shares to reconstruct the message. In Section 5, we show an advanced construction that reduces the overhead to $O(t_a)$ or even $O(1)$ assuming PRG.

Supporting BC Messages. In the synchronous setting, when $t_s \geq n/3$, it is known that guaranteed output delivery is only possible assuming a broadcast channel. Thus, we further extend $\mathcal{F}_{\mathsf{Channel}}$ to support broadcasting messages for each party A. Similar to P2P messages, we want to ensure that either (1) A successfully broadcasts his message m, or (2) all parties identify the party A, which is guaranteed to be corrupted if the network is synchronous (but might be an honest party if the network is asynchronous).

Protocol $\Pi_{\texttt{BC-Send}}(A, \texttt{rid})$

1: After A enters the `rid`-th round, A broadcasts his message m to all parties.
2: Recall that $\texttt{Time}_{\texttt{bc}}$ is the time required by $\Pi_{\texttt{RBC}}$ when the dealer is honest and the network is synchronous. All parties invoke $\Pi_{\texttt{BA}}$ and decide their input as follows: After entering the `rid`-th round, each party P_i waits for $\Delta + \texttt{Time}_{\texttt{bc}}$ for the broadcast message from A.
   – If P_i receives the broadcast message from A within the time limit, P_i sets his input of $\Pi_{\texttt{BA}}$ to be 1.
   – Otherwise, P_i sets his input of $\Pi_{\texttt{BA}}$ to be 0.
   If $\Pi_{\texttt{BA}}$ outputs 1, all parties output the broadcast message m from A. Otherwise, all parties output $(\texttt{Eliminate}, A)$.

We show that when the network is synchronous, an honest party A will not be identified as a corrupted party. It is sufficient to show that in this case, an honest P_i will set his input of $\Pi_{\texttt{BA}}$ to be 1: After P_i enters the `rid`-th round, A is guaranteed to enter the `rid`-th round within Δ time. Then P_i is guaranteed to receive the broadcast message from A within another $\texttt{Time}_{\texttt{bc}}$.

When the network is asynchronous, termination for honest parties follows from the termination of $\Pi_{\texttt{BA}}$ to which all honest parties input their bit at time $\Delta + \texttt{Time}_{\texttt{bc}}$. Similar to the argument for $\Pi_{\texttt{P2P-Send}}$, if $\Pi_{\texttt{BA}}$ outputs 1, then all honest parties will eventually receive the broadcast message from A.

Summary. To realize $\mathcal{F}_{\mathsf{Channel}}$, we simply invoke $\Pi_{\texttt{P2P-Send}}$ for every pair of parties and invoke $\Pi_{\texttt{BC-Send}}$ for every party. To be more concrete,

1. Each party P_i receives $(\texttt{rid}, m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ as input where `rid` is the round identifier, $m_0^{(i)}$ is the message to be broadcast, and $m_j^{(i)}$ is the message to be sent to P_j.
2. All parties invoke $\Pi_{\texttt{P2P-Send}}(P_i, P_j, \texttt{rid})$ to let P_j send his `rid`-th round message $m_i^{(j)}$ to P_i.
3. All parties invoke $\Pi_{\texttt{BC-Send}}(P_i, \texttt{rid})$ to let P_i broadcast his `rid`-th round message $m_0^{(i)}$ to all parties.
4. All parties decide the set W as follows.
   – For each call of $\Pi_{\texttt{P2P-Send}}(P_i, P_j, \texttt{rid})$, if the output is $(\texttt{Eliminate}, P_i)$, all parties insert P_i in W. If the output is $(\texttt{Eliminate}, P_j)$, all parties insert P_j in W.
   – For each call of $\Pi_{\texttt{BC-Send}}(P_i, \texttt{rid})$, if the output is $(\texttt{Eliminate}, P_i)$, all parties insert P_i in W.
5. For each party $P_i \in W$, P_i sets his output to be $(\texttt{rid}, W, \varnothing)$. For each party $P_i \notin W$, P_i sets his output to be $(\texttt{rid}, W, \{(m_0^{(j)}, m_i^{(j)})\}_{i \notin W})$.

In Section 5, we give the formal descriptions of $\mathcal{F}_{\mathsf{Channel}}$ and our construction, and prove its security.

2.3 Overview of Our Compiler

With $\mathcal{F}_{\mathsf{Channel}}$, we are ready to give a more detailed description of our compiler.

An Initial Attempt. We assume a GOD protocol $\Pi_{\texttt{SMPC-GOD}}$ that is secure against t_s corrupted parties in the synchronous setting and a GOD protocol $\Pi_{\texttt{AMPC-GOD}}$ that is secure against $1/3$ corruption in the asynchronous setting. All parties initialize a set $W = \varnothing$ at the beginning. Suppose R is the (upper bound of the) number of rounds required by $\Pi_{\texttt{SMPC-GOD}}$. From $\texttt{rid} = 1$ to R, all parties execute the following steps to emulate the $\texttt{rid}$-th round of $\Pi_{\texttt{SMPC-GOD}}$.

1. For each party $P_i \in W$, he sets $m_j^{(i)} = \perp$ for all $j \in \{0, \ldots, n\}$. For each party $P_i \notin W$, he follows $\Pi_{\texttt{SMPC-GOD}}$ to compute the broadcast message $m_0^{(i)}$ and each P2P message $m_j^{(i)}$ that is supposed to be sent to P_j for all $j \in \{1, \ldots, n\}$.
2. All parties invoke $\mathcal{F}_{\mathsf{Channel}}$ with inputs $(\texttt{rid}, m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$. Let W' be the set of new identified parties from $\mathcal{F}_{\mathsf{Channel}}$. Then for each party $P_i \in W'$, P_i receives $(\texttt{rid}, W', \varnothing)$. For each party $P_i \notin W'$, P_i receives $(\texttt{rid}, W, \{(m_0^{(j)}, m_i^{(j)})\}_{i \notin W})$.
3. All parties set $W = W \cup W'$ and decide their messages received in the $\texttt{rid}$-th round as follows.
   - For each $P_i \in W$, P_i takes all-0 strings as messages he received in the $\texttt{rid}$-th round.
   - For each $P_i \notin W$, P_i uses $m_0^{(j)}, m_i^{(j)}$ as the broadcast message and P2P message from P_j for each $P_j \notin W$, and uses all-0 strings as the messages from P_j for each $P_j \in W$.
4. Depending on the size of W and $\texttt{rid}$, all parties run the following steps.
   - If $|W| \leq t_s - t_a$ and $\texttt{rid} = R$, P_i computes his output following $\Pi_{\texttt{SMPC-GOD}}$.
   - If $|W| \leq t_s - t_a$ and $\texttt{rid} < R$, P_i moves to the $(\texttt{rid}+1)$-th round of $\Pi_{\texttt{SMPC-GOD}}$.
   - Otherwise, all parties take the first $t_s - t_a$ parties in W as the output and terminates.

Recall that in $\mathcal{F}_{\mathsf{Channel}}$, we assume that when the network is synchronous, for each round of $\Pi_{\texttt{SMPC-GOD}}$, all honest parties enter the round at a similar time. We show that this is indeed the case in the above sketch. First, this assumption is true for the first round. For each $\texttt{rid} \geq 2$, an honest party enters the $\texttt{rid}$-th round after he terminates $\Pi_{\texttt{RBC}}$ and $\Pi_{\texttt{BA}}$ in all instances of $\Pi_{\texttt{P2P-Send}}$ and $\Pi_{\texttt{BC-Send}}$ invoked for the $(\texttt{rid}-1)$-th round. By the property of $\Pi_{\texttt{RBC}}$ and $\Pi_{\texttt{BA}}$, every other honest party will terminate these $\Pi_{\texttt{RBC}}$ and $\Pi_{\texttt{BA}}$ and thus enter the $\texttt{rid}$-th round within Δ time.

As we discussed above, after identifying $t_s - t_a$ parties which are guaranteed to be corrupted if the network is synchronous,

- if the network is synchronous, there are at most $t_s - (t_s - t_a) = t_a$ corrupted parties in the remaining $n - (t_s - t_a) = t_s + 2t_a + 1$ parties;
- if the network is asynchronous, there are at most t_a corrupted parties in the remaining $n - (t_s - t_a)$ parties.

In either case, the corruption threshold becomes $1/3$ and we can fallback to an asynchronous MPC protocol securely against $1/3$ corruption.

Security Issues. Unfortunately, the above construction does not yield a secure network-agnostic MPC protocol. The main issue is that the required security of $\Pi_{\texttt{SMPC-GOD}}$ does not match the standard adaptive security as we elaborate below.

Recall that when the network is asynchronous, the identified parties can be honest. In our compiler, identified parties are viewed as corrupted parties and all parties just take all-0 strings as messages from identified parties. This means that when the network is asynchronous, an adversary may *adaptively* crash up to $t_s - t_a$ honest parties. Thus, we need $\Pi_{\texttt{SMPC-GOD}}$ to be secure against the following kind of adversaries:

- The adversary may corrupt a subset of $t \leq t_s$ parties in the beginning of the computation. The adversary will have the full control of the behavior of these parties.
- During the computation, the adversary can adaptively let up to $t_s - t$ honest parties crash. The adversary does *not* have the control of these parties.

Note that by crashing an honest party, the adversary *does not learn* the view, and in particular the input, of this honest party. This is unlike the standard adaptive security where when an honest party is corrupted, the adversary will learn its view and control its behavior.

At a first glance, our required security may sound weaker than the standard adaptive security since the adversary in our case is strictly weaker than that in the standard adaptive security (not being able to access the view and control the behavior of crashed honest parties). Unfortunately, this is not true. The reason is that in our case the ideal adversary (simulator) is also weaker: when crashing an honest party, the ideal adversary does not learn its input. In comparison, in the standard adaptive security, the ideal adversary can learn the input of an honest party when this party becomes corrupted.

We note that this is not an artificial issue. A common strategy of building MPC protocol with GOD is to use party elimination (or dispute control). When a corrupted party is identified, his input can be reconstructed without breaking the security and this trick has been used in [BSFO12] and [GSZ20]. This, however, would break the security in our case since the party being identified can be a crashed honest party.

Another issue is that in our compiler, the emulation of $\Pi_{\texttt{SMPC-GOD}}$ may fail in the last round. In this case, the adversary may learn the function output but prevent honest parties from learning the function output. Then when falling back to the asynchronous MPC, the adversary can switch to a different set of inputs for corrupted parties, thus breaking the security.

Resolving Security Issues Relying on Robust Secret Sharings. To resolve the above two issues, our idea is to make use of a robust secret sharing (RSS) scheme. Let $[\![s]\!]$ denote a robust secret sharing of the secret s. Very informally, it satisfies that (1) any t_s shares are independent of the secret s, and (2) the secret s can always be reconstructed as long as the number of incorrect shares (from corrupted parties) are bounded by t_s. Such a robust secret sharing scheme can be built with unconditional security for all $t_s < n/2$ [BSFO12].

Recall that the gap between our required security and the standard adaptive security is that when an honest party is crashed in our case, the (ideal) adversary does not learn its input or output. We observe that such a gap can be overcome if either (1) the crashed honest party does not have input/output, or (2) the crashed honest party's input/output are secret shares. In the former case, the view of this crashed honest party can be simulated in the standard adaptive security without requiring any further information. In the latter case, since the total number of crashed honest parties plus the number of corrupted parties is bounded by t_s, by the security of robust secret sharing schemes, this honest party's input/output can be simulated without learning the secrets.

Thus, our construction works in three phases.

Input Phase. We first let each party (acting as the dealer) secret share his input via a robust secret sharing scheme. To be more concrete, we apply our compiler on a synchronous MPC with GOD that securely computes the input sharing functionality with adaptive security.

In this case, only the dealer has input and other parties only have outputs which are secret shares. Thus if the crashed honest party is not the dealer, we may simulate this honest party's share without learning the secret. To deal with the case where the crashed honest party is the dealer, we modify our compiler by letting all parties abort the input sharing protocol when the dealer is crashed and assuming the input sharing protocol fails for this dealer. Intuitively, what an adversary $\mathcal{A}$ can learn in this case is subsumed by the case where $\mathcal{A}$ never crashes the dealer. We show that this can be reduced to an adversary in the standard adaptive security.

Computation Phase. We then apply our compiler on a synchronous MPC with GOD that securely computes from the secret-shared inputs to secret-shared outputs. In this case, all parties' inputs and outputs are secret shares. Following our observation, when an honest party crashed, we may simulate this party's shares without learning the secrets, and reduce the security to the standard adaptive security of the underlying protocol.

Output Phase. Finally, with robust secret sharing of each output value, all parties can reconstruct the outputs. Note that this also addresses the second issue since corrupted parties cannot stop honest parties from reconstructing the outputs at this step.

Fallback Procedure. When a set W of $t_s - t_a$ parties are identified, all parties move to the fallback procedure: Parties that are not in W run an asynchronous MPC with GOD to compute the function output.

Instantiations of Building Blocks. To obtain a network-agnostic MPC only assuming a setup for signatures, we instantiate the building blocks required by our compiler. For robust secret sharing schemes, we follow the technique in [BDOZ11, BFO12] by using pairwise information-theoretic MACs, which is further used in [DEP23]. The sharing size is $O(n^2\kappa)$ bits. See Section C for more details. For synchronous MPCs, we use the protocols in [BFO12, GSZ20] which can achieve linear communication in the number of parties. For the asynchronous MPC, we use the protocols in [GLZS24, JLS24] that also achieve linear communication. We refer the readers to Section E.2 for more details.

We note that our compiler can potentially achieve linear communication also in the information-theoretic setting when using a better information-theoretic MPC protocol in the synchronous setting. Recall that in our compiler, it is sufficient to use a synchronous MPC that is secure against t_s corruptions. On the other hand, the instantiations from [BSFO12, GSZ20] are secure in the standard honest majority setting where $t = (n-1)/2$. In the strong honest majority setting (where there is a gap between the corruption threshold t_s and $n/2$), one can potentially use the packed Shamir secret sharing scheme to further bring down the communication complexity by a factor of $n/2 - t_s = O(t_a)$. This is indeed the case for malicious security with abort [GPS21, GPS22], while the GOD case remains unclear. However, we expect that full security is possible when combining with techniques in [BSFO12, GSZ20].

3 Model

We consider the universal composability (UC) framework [Can01] to prove the security of our protocols. Note that the standard UC framework considers an asynchronous network model where the adversary is allowed to drop messages. We are interested in 1) the synchronous model where parties have access to synchronized clocks and messages are delivered within a publicly known delay, and 2) the asynchronous setting with *eventual delivery*, where messages sent by honest parties are eventually delivered. To capture such models, several variants have been proposed in the literature for synchronous UC [KMTZ13, Can01, LZLM+19, BDD+21] where functionalities keep track of time, and asynchronous UC with eventual delivery [CGHZ16, LZLM+19, CFG+23]. Our protocols can be proven secure in any of those models. When modeling functionalities, we will use the term *request-based delayed output* [Coh16, CP23] to refer to eventual output delivery. Moreover, for functionalities in the synchronous setting, we use the term *time-bound delayed output* to ensure that the output is delivered within a prescribed time-bound, but the adversary may deliver the message earlier.

3.1 Ideal World

We consider the setting of multi-party computation with guaranteed output delivery (GOD). In this setting, we model the ideal MPC functionality with a flag mode, indicating whether the setting is synchronous or asynchronous. When the network is synchronous, the functionality expects the inputs by a certain time and will take into account all inputs from honest parties for the computation, and deliver the outputs by a certain predetermined time. When the network is asynchronous, only $n - t_s - t_a$ honest inputs are guaranteed to be taken into account, which is optimal as proven in [BZL20], and the outputs are eventually delivered. We refer the readers to Section A for the ideal functionality.

3.2 Real World

Network Model. In our protocols, parties have access to a complete network of secure point-to-point channels. Our protocols operate in two possible settings: synchronous or asynchronous.

In the synchronous setting, all parties have access to synchronized clocks and all messages are guaranteed to be delivered within some known upper bound delay Δ. Within Δ, the adversary can schedule the messages arbitrarily. In particular, the adversary is *rushing*, i.e., within the same round, the adversary is allowed to send its messages after seeing the honest parties' messages. Sometimes it is convenient to describe a protocol

in rounds, where each round r refers to the interval of time $(r-1)\Delta$ to $r\Delta$. In such case, we say that a party receives a message in round r if it receives the message within that time interval. Moreover, we say a party sends a message in round r when it sends the message at the beginning of the round, i.e., at time $(r-1)\Delta$.

In the asynchronous setting, both assumptions above are removed. That is, parties do not have access to synchronized clocks (but they advance), and the adversary is allowed to arbitrarily schedule the delivery of the messages. However, we assume that all messages are eventually delivered (i.e., the adversary cannot drop messages).

Adversary Model. We consider a static adversary who corrupts parties in an arbitrary manner at the beginning of the protocol, i.e. active corruption. In the synchronous setting, the adversary can corrupt up to t_s parties. In the asynchronous setting, the adversary can corrupt up to t_a parties.

4 Preliminary

4.1 Signatures

Our construction makes use of a signature scheme. Recall that a signature scheme allows a party P_r to verify the owner and the integrity of a message m received from a sender P_s. We give two instantiations of signature schemes, one with information-theoretic security, and the other one with computational security.

Signature Scheme with Information-Theoretic Security. We first give the syntax of an unconditionally secure signature scheme, which is also referred to as a pseudo-signature, following the definition in [PW96, SHZI02]. Here the key generation algorithm needs to take as input a set $\mathcal{P}$ and two numbers ℓ_s, ℓ_v, where the set $\mathcal{P}$ includes the identities of verifiers who can verify the signatures later ($\mathcal{P}$-verifiable) and the numbers ℓ_s, ℓ_v denote the upper bounds of the signatures that can be generated and verified. To achieve information-theoretic security, each party's verification key cannot be publicly known. That means the signer has a signing key sk and each party $P_i \in \mathcal{P}$ holds a different private verification key vk_i corresponding to sk.

Definition 1. *[PW96, SHZI02] A $\mathcal{P}$-verifiable signature scheme Π consists of three randomized algorithms* $(\mathtt{Gen}, \mathtt{Sign}, \mathtt{Vrfy})$ *where*

1. ***Key Generation:*** $\mathtt{Gen}$ *takes security parameter κ, a set $\mathcal{P}$ and two numbers $\ell_s, \ell_v \in \mathbb{N}$ as inputs and outputs a pair* $(\mathsf{sk}, \mathbf{vk})$*, where* $\mathsf{sk} \in \{0,1\}^\kappa$ *is a signing key,* $\mathbf{vk} = \{\mathsf{vk}_i\}_{i\in\mathcal{P}}$ *and each* $\mathsf{vk}_i \in \{0,1\}^\kappa$ *is a verification key corresponding to* sk*.*
2. ***Signature Generation:*** $\mathtt{Sign}$ *takes the message m and the signing key* sk *as inputs and outputs a signature $\sigma \in \{0,1\}^{poly(\kappa)}$.*
3. ***Verification:*** $\mathtt{Vrfy}$ *takes the message m, a signature σ and a verification key* vk_i *as inputs and output a decision-bit $b \in \{0,1\}$.*

We state the security requirement and analyze the cost of an unconditionally secure signature scheme in Section B.1.

Computationally Secure Signature Scheme. We briefly recall the syntax of a standard signature scheme.

Definition 2. *A computational signature scheme Σ is a triple of efficient algorithms $\Sigma = (\mathtt{Gen}, \mathtt{Sign}, \mathtt{Vrfy})$, with the following syntax:*

1. ***Key Generation:*** $\mathtt{Gen}$ *takes security parameter κ and outputs a pair* $(\mathsf{sk}, \mathsf{vk})$*, where* $\mathsf{sk} \in \{0,1\}^\kappa$ *is the signing key, and* $\mathsf{vk} \in \{0,1\}^\kappa$ *is the verification key.*
2. ***Signature Generation:*** $\mathtt{Sign}$ *takes the message m and the signing key* sk *as inputs and outputs a signature σ.*
3. ***Verification:*** $\mathtt{Vrfy}$ *takes the message m, a signature σ and the verification key* vk *as inputs and output a decision-bit $b \in \{0,1\}$.*

We require signatures to satisfy correctness, meaning that $\texttt{Vrfy}(m, \texttt{Sign}(m, \mathsf{sk}), \mathsf{vk}) = 1$ *for all* $m \in \{0,1\}^*$ *and for* $(\mathsf{sk}, \mathsf{vk}) = \texttt{Gen}(1^\kappa)$*. We also require our signatures to satisfy the standard unforgeability against chosen message attacks [GMR88].*

In our computational protocols we make use of any signature scheme with keys of size $\mathcal{O}(\kappa)$ and signatures of size $\mathcal{O}(\kappa)$ per field element message [ElG85, Sch90, JMV01, BLS01].

4.2 Network Agnostic Byzantine Agreement

In a byzantine agreement (BA) protocol, each party P_i takes a message $m_i \in \mathcal{M}$ as input, where $\mathcal{M}$ is the message space which contains $\perp$. The protocol allows all parties to agree on a common message in $\mathcal{M}$. We recall the property-based definition of a BA protocol as follows.

Definition 3. *Let* t *be the corruption threshold and* Π *be a protocol executed by parties* $\mathcal{P} = \{P_1, \ldots, P_n\}$*, where each party initially holds an input* $m_i \in \mathcal{M}$*, and all parties terminate upon getting output.*

- t***-Validity:*** *If all honest parties have the same input* $m \in \mathcal{M}$*, then all honest parties take* m *as output.*
- t***-Consistency:*** *Honest parties do not obtain different outputs in* $\mathcal{M}$*.*
- t***-Termination:*** *Suppose all honest parties participate in the protocol with their inputs in* $\mathcal{M}$*, and they never stop participating until termination. Then all honest parties terminate with a valid output in* $\mathcal{M}$*.*

For a BA protocol Π, if Π achieves t-validity, t-consistency, and t-termination, we say Π is t-secure. We say Π is a network agnostic BA protocol if Π is t_s-secure in the synchronous setting and t_a-secure in the asynchronous setting. When the network is synchronous, the t_s-termination property guarantees that all honest parties would terminate Π within a deterministic delay, which depends on their starting time. We use $\texttt{Time}_{\texttt{ba}}$ to denote the running time when the starting time of all honest parties differ by at most Δ.

In this work, we require all honest parties to terminate the BA protocol at a similar time in the synchronous network. We refer the reader to Section B.2 to see how this is achieved, and the cost in different settings.

4.3 Network Agnostic Reliably Broadcast

In a reliably broadcast (RBC) protocol, a sender P_s takes a message $m \in \mathcal{M}$ as input, where $\mathcal{M}$ is the message space which contains $\perp$. The protocol allows the sender to consistently distribute the message m among a set of parties. We recall the property-based definition of RBC protocol as follows.

Definition 4. *Let* t *be the corruption threshold and* Π *be a protocol executed by parties* $\mathcal{P} = \{P_1, \ldots, P_n\}$*, where a sender* $P_s \in \mathcal{P}$ *holds an input* $m \in \mathcal{M}$*, and all parties terminate upon getting output.*

- t***-Validity:*** *If the sender* P_s *is honest, then every honest parties output* m*.*
- t***-Consistency:*** *If an honest party terminates, then his output is a valid message* $m' \in \mathcal{M}$*, and all honest parties will eventually terminate with output* m'*.*

If Π achieves t-validity and t-consistency, we say that Π is t-secure. A network agnostic RBC protocol is t_s-secure in the synchronous setting and t_a-secure in the asynchronous setting. Based on a network agnostic BA protocol $\Pi_{\texttt{BA}}$, we can realize RBC protocol by first letting the sender P_s send the message m to all parties, then all parties invoke $\Pi_{\texttt{BA}}$ with the message received from the sender, and finally terminate with the output of $\Pi_{\texttt{BA}}$. We denote this RBC protocol as $\Pi_{\texttt{RBC}}$ and we can also ensure that all honest parties terminate at a similar time in the synchronous network. The asymptotic cost of $\Pi_{\texttt{RBC}}$ is identical to that of $\Pi_{\texttt{BA}}$ in Section 4.2. We use $\texttt{Time}_{\texttt{bc}}$ to denote the running time of $\Pi_{\texttt{RBC}}$ when the dealer is honest. Note that if the dealer is honest, then all honest parties will receive the broadcast message within Δ time, which ensures that the starting time of the underlying $\Pi_{\texttt{BA}}$ of all honest parties differ by at most Δ.

4.4 Robust Secret Sharing

Informally, a (t, n)-robust secret sharing scheme allows a dealer D to share a secret s to all parties such that any t shares are independent of the secret. When needed, all parties can together reconstruct the secret s even with t incorrect shares (provided by corrupted parties). Our construction further requires the robust secret sharing scheme to achieve adaptive secrecy, online reconstruction, and efficient resampling.

Informally, the adaptive secrecy allows the adversary to learn t shares chosen adaptively while still preserving the privacy of the secret. Online reconstruction allows the receiver to reconstruct the secret even if the network is asynchronous (where honest parties' shares come one by one with unknown delay and corrupted parties may not send their shares). Efficient resampling allows us to resample the shares of honest parties given the shares of corrupted parties and the secret. We refer the readers to Section B.3 for the formal definition and the instantiation.

5 Secure Channel with Synchronous Identifiable Abort

In this section, we give the formal description of the functionality $\mathcal{F}_{\mathsf{Channel}}$. Recall that we use n for the number of parties, t_s for the synchronous corruption threshold, and t_a for the asynchronous corruption threshold. We further assume that $t_a \leq t_s$ and $n = 2t_s + t_a + 1$.

In the beginning, the functionality $\mathcal{F}_{\mathsf{Channel}}$ takes as input two parameters Δ and τ_{synch}, where Δ is the delivery time-bound for a message sent in the synchronous network and τ_{synch} is the time-bound for giving outputs of honest parties in the synchronous mode. The functionality $\mathcal{F}_{\mathsf{Channel}}$ runs in two different modes. In the synchronous mode, we assume that all honest parties enter each round at a similar time (with time difference bounded by Δ). And all parties will move on to the next round only after the current round is finished. In each round, $\mathcal{F}_{\mathsf{Channel}}$ first records the messages $(m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ to be sent by each party P_i. Here $m_0^{(i)}$ represents the broadcast message, and $m_j^{(i)}$ represents the message to be sent to P_j. Before the timeout τ_{synch}, the adversary may choose a subset W of parties in $\mathcal{C}orr$ and $\mathcal{F}_{\mathsf{Channel}}$ will distribute messages to each party that is not in W. The set W represents the identified parties in this round. The functionality $\mathcal{F}_{\mathsf{Channel}}$ guarantees that if an honest party terminates the current round, then all other honest parties terminate the current round within Δ time. In the asynchronous mode, parties may enter each round at a different time. In this case, we allow the adversary $\mathcal{S}$ to choose any subset W of parties in $\mathcal{P}$.

Functionality $\mathcal{F}_{\mathsf{Channel}}$

$\mathcal{F}_{\mathsf{Channel}}$ is parameterized by a set $\mathcal{P} = \{P_1, \ldots, P_n\}$, an adversary $\mathcal{S}$, corruption threshold t_s, t_a, and a variable $\mathsf{mode} \in \{\mathsf{synch}, \mathsf{asynch}\}$. $\mathcal{F}_{\mathsf{Channel}}$ receives the delivery time-bound Δ for a message sent in the synchronous network. In addition, $\mathcal{F}_{\mathsf{Channel}}$ has timeout τ_{synch} for giving outputs of honest parties in the synchronous mode. $\mathcal{F}_{\mathsf{Channel}}$ receives a set $\mathcal{R}$ of valid round ids.

Initialization

1: $\mathcal{F}_{\mathsf{Channel}}$ receives the set $\mathcal{C}orr$ of corrupted parties from $\mathcal{S}$ and the network configuration $\mathsf{mode} \in \{\mathsf{synch}, \mathsf{asynch}\}$. $\mathcal{F}_{\mathsf{Channel}}$ checks that $|\mathcal{C}orr| \leq t_s$ if $\mathsf{mode} = \mathsf{synch}$ and $|\mathcal{C}orr| \leq t_a$ if $\mathsf{mode} = \mathsf{asynch}$. If not, $\mathcal{F}_{\mathsf{Channel}}$ gives up the security[a].

2: $\mathcal{F}_{\mathsf{Channel}}$ initiates a working round id $\mathtt{wID} = \perp$ and a set of used round ids $\mathcal{U} = \varnothing$.

[a] By giving up the security, $\mathcal{F}_{\mathsf{Channel}}$ sends its view to $\mathcal{S}$ and allows $\mathcal{S}$ to arbitrarily decide the output of honest parties and the delivery time.

Functionality $\mathcal{F}_{\mathsf{Channel}}$

Synchronous: $\mathsf{mode} = \mathsf{synch}$

1: Upon receiving a request $(\texttt{rid}, m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ from P_i such that $\texttt{rid} \in \mathcal{R}$, $\mathcal{F}_{\mathsf{Channel}}$ records the request and ignores all other requests starting with $\texttt{rid}$ from P_i. Furthermore, if P_i is honest, $\mathcal{F}_{\mathsf{Channel}}$ does the following: $\mathcal{F}_{\mathsf{Channel}}$ sends $(P_i, \texttt{rid}, m_0^{(i)}, \{m_j^{(i)}\}_{j \in \mathcal{C}orr})$ and its receiving time τ to $\mathcal{S}$. Then
   - If $\texttt{wID} = \perp$ and $\texttt{rid} \notin \mathcal{U}$, $\mathcal{F}_{\mathsf{Channel}}$ sets $\texttt{wID} = \texttt{rid}$, records its receiving time $\tau_{\texttt{rid}} := \tau$, and adds $\texttt{rid}$ to $\mathcal{U}$.
   - If $\texttt{wID} = \texttt{rid}$ and $\tau \leq \tau_{\texttt{rid}} + \Delta$, $\mathcal{F}_{\mathsf{Channel}}$ continues.
   - In all other cases, $\mathcal{F}_{\mathsf{Channel}}$ gives up the security.

2: Upon reaching time $\tau_{\texttt{rid}} + \Delta$, $\mathcal{F}_{\mathsf{Channel}}$ checks whether he has recorded $(\texttt{rid}, m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ for every honest party P_i. If not, $\mathcal{F}_{\mathsf{Channel}}$ gives up the security.

3: Upon receiving $(\mathsf{Proceed}, \texttt{rid}, W)$ from $\mathcal{S}$ at time τ such that $\texttt{rid} \in \mathcal{R}$, $\mathcal{F}_{\mathsf{Channel}}$ checks that (1) $\texttt{wID} = \texttt{rid}$, (2) $\tau \leq \tau_{\texttt{rid}} + \tau_{\mathsf{synch}}$, (3) $W \subset \mathcal{C}orr$, and (4) for each $P_i \notin W$, $\mathcal{F}_{\mathsf{Channel}}$ has received the request $(\texttt{rid}, m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ from P_i. If not, $\mathcal{F}_{\mathsf{Channel}}$ ignores this message. Otherwise, $\mathcal{F}_{\mathsf{Channel}}$ sets $\texttt{wID} = \perp$ and for each honest P_j, $\mathcal{F}_{\mathsf{Channel}}$ sends a time-bound delayed output $(\texttt{rid}, W, \{(i, m_0^{(i)}, m_j^{(i)})\}_{i \notin W})$ to P_j with time-bound $\tau + \Delta$.

   Here a time-bound delayed output ensures that the functionality $\mathcal{F}_{\mathsf{Channel}}$ will deliver the message within the time-bound, while the adversary $\mathcal{S}$ may ask to deliver the message before the time-bound.

4: Upon reaching time $\tau_{\texttt{rid}} + \tau_{\mathsf{synch}}$, if $\texttt{wID} \neq \texttt{rid}$, $\mathcal{F}_{\mathsf{Channel}}$ continues. Otherwise, $\mathcal{F}_{\mathsf{Channel}}$ checks that for each $P_i \notin \mathcal{C}orr$, $\mathcal{F}_{\mathsf{Channel}}$ has received the request $(\texttt{rid}, m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ from P_i.
   - If true, $\mathcal{F}_{\mathsf{Channel}}$ sets $\texttt{wID} = \perp$, and for each honest P_j, $\mathcal{F}_{\mathsf{Channel}}$ sends $(\texttt{rid}, \mathcal{C}orr, \{(i, m_0^{(i)}, m_j^{(i)})\}_{i \notin \mathcal{C}orr})$ to P_j.
   - Otherwise, $\mathcal{F}_{\mathsf{Channel}}$ gives up the security.

Functionality $\mathcal{F}_{\mathsf{Channel}}$

Asynchronous: mode = asynch

1: Upon receiving a request $(\texttt{rid}, m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ from P_i such that $\texttt{rid} \in \mathcal{R}$, $\mathcal{F}_{\mathsf{Channel}}$ records the request and ignores all other requests starting with $\texttt{rid}$ from P_i. Furthermore, if P_i is an honest party, $\mathcal{F}_{\mathsf{Channel}}$ does the following: $\mathcal{F}_{\mathsf{Channel}}$ sends $(P_i, \texttt{rid}, m_0^{(i)}, \{m_j^{(i)}\}_{j \in \mathcal{C}orr})$ to $\mathcal{S}$. If this is the first request starting with $\texttt{rid}$ from an honest party, for each honest party P_j, $\mathcal{F}_{\mathsf{Channel}}$ sends a request-based delayed output $(\texttt{rid}, \mathcal{P}, \varnothing)$ to P_j.

2: Upon receiving $(\mathsf{Proceed}, \texttt{rid}, W)$ from $\mathcal{S}$ such that $\texttt{rid} \in \mathcal{R}$, $\mathcal{F}_{\mathsf{Channel}}$ checks that (1) $\texttt{rid} \notin \mathcal{U}$, (2) for each $P_i \notin W$, $\mathcal{F}_{\mathsf{Channel}}$ has received the request $(\texttt{rid}, m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ from P_i, and (3) $(\texttt{rid}, \mathcal{P}, \varnothing)$ has not been delivered to any honest party. If not, $\mathcal{F}_{\mathsf{Channel}}$ ignores this message. Otherwise, $\mathcal{F}_{\mathsf{Channel}}$ adds $\texttt{rid}$ in $\mathcal{U}$. Then for each honest $P_j \in W$, $\mathcal{F}_{\mathsf{Channel}}$ replaces the request-based delayed output $(\texttt{rid}, \mathcal{P}, \varnothing)$ for P_j by $(\texttt{rid}, W, \varnothing)$. For each honest $P_j \notin W$, $\mathcal{F}_{\mathsf{Channel}}$ replaces the request-based delayed output $(\texttt{rid}, \mathcal{P}, \varnothing)$ for P_j by $(\texttt{rid}, W, \{(i, m_0^{(i)}, m_j^{(i)})\}_{i \notin W})$.

5.1 Realizing $\Pi_{\texttt{P2P-Send}}$ and $\Pi_{\texttt{BC-Send}}$

To realize $\mathcal{F}_{\mathsf{Channel}}$, we first focus on a P2P message from a sender B to a receiver A. Following the discussion in Section 2.2, we assume that when the network is synchronous, for each round, all honest parties enter this round at a similar time. To be more concrete, we assume that the time difference between the starting time of any two honest parties is bounded by Δ. Recall that Δ is the delivery time-bound for a message sent in the synchronous network.

Our construction assumes a signature scheme and makes use of a network-agnostic broadcast protocol $\Pi_{\texttt{RBC}}$ and a network-agnostic agreement protocol $\Pi_{\texttt{BA}}$ with the following property:

- When the network is synchronous, if an honest party terminates $\Pi_{\texttt{RBC}}$ (or $\Pi_{\texttt{BA}}$), then all honest parties terminate $\Pi_{\texttt{RBC}}$ (or $\Pi_{\texttt{BA}}$) within Δ time.

We use $\texttt{Time}_{\texttt{bc}}$ for the running time of $\Pi_{\texttt{RBC}}$ when the network is synchronous and the dealer is honest, and $\texttt{Time}_{\texttt{ba}}$ for the running time of $\Pi_{\texttt{BA}}$ when the network is synchronous and the starting time of all honest

parties differs by Δ at most. At the beginning of the execution, we assume a trusted setup for signatures as described in $\mathcal{F}_{\mathsf{Setup}}$ (Section 2.2).

In the following, we directly present our construction with $O(t_a)$ multiplicative overhead. The initial sending attempt remains unchanged (See Section 2.2 for more details).

Advanced Resend Attempt. At a high level, when the initial attempt fails, the receiver A will propose a set of t_a+1 parties which are not disputed with either A or B as the intermediates that pass the shares from B to A. If the resend attempt fails, A will identify new parties that are either disputed with A or B. Then A will repeat the above process. Note that across the whole execution, the resend attempt will fail at most n times. This ensures that the amortized communication overhead of simulating the P2P channel is $O(t_a)$.

Protocol $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$

Advanced Resend Attempt

The receiver A maintains two sets of parties D_A and D_B, which contain parties that have dispute with A and B respectively. A also maintains the set `Proof` which contains $(\texttt{Dispute}, P_i, B)$ with a valid signature from P_i for each $P_i \in D_B$. A maintains a counter `ReCount` representing the number of resend attempts for B's messages, which is set to 0 in the beginning of the whole execution. Let R be the (upper bound of the) number of rounds. A repeats the following steps until one of the following three conditions is satisfied: (1) A receives t_a+1 message shares; (2) $|D_A| \geq t_s + 1$; (3) $|D_B| \geq t_s + 1$.

1: The receiver A updates $\texttt{ReCount} := \texttt{ReCount} + 1$. Then A chooses the set S of the first $t_a + 1$ parties that are not in $D_A \cup D_B$ and broadcasts $(\texttt{Resend}, A, B, \texttt{ReCount}, \texttt{rid}, S)$. Here we use $(\texttt{Resend}, A, B, \texttt{ReCount})$ as the unique identifier for the broadcast message $(\texttt{Resend}, A, B, \texttt{ReCount}, \texttt{rid}, S)$. We require that all parties only participate $\Pi_{\texttt{RBC}}$ with identifier $(\texttt{Resend}, A, B, \texttt{ReCount})$ when $1 \leq \texttt{ReCount} \leq n + R$.

2: The sender B samples a random degree-t_a Shamir sharing of his message m, denoted by $[m]_{t_a} := (m_1, \ldots, m_n)$. This step only runs once.

3: Each party P_i, upon receiving $(\texttt{Resend}, A, B, \texttt{ReCount}, \texttt{rid}, S)$ broadcast by A, if $|S| = t_a + 1$, P_i accepts and records the request. Otherwise, P_i rejects the request.

4: The sender B, after entering the `rid`-th round and accepting $(\texttt{Resend}, A, B, \texttt{ReCount}, \texttt{rid}, S)$, sends $(\texttt{Resend}, A, B, \texttt{rid}, m_i)$ together with his signature to each $P_i \in S$.

5: Each party P_i, after entering the `rid`-th round and accepting $(\texttt{Resend}, A, B, \texttt{ReCount}, \texttt{rid}, S)$, if P_i receives (or has received) $(\texttt{Resend}, A, B, \texttt{rid}, m_i)$ with a valid signature from B within 2Δ time, P_i sends $(\texttt{Resend}, A, B, \texttt{rid}, m_i)$ and the signature of B to A. Otherwise, P_i sends $(\texttt{Dispute}, P_i, B)$ together with his signature to A.

6: Recall that $\texttt{Time}_{\texttt{bc}}$ is the time required by $\Pi_{\texttt{RBC}}$ when the dealer is honest and the network is synchronous. The receiver A, after starting $\Pi_{\texttt{RBC}}$ for $(\texttt{Resend}, A, B, \texttt{ReCount}, \texttt{rid}, S)$, waits for the response from each party $P_i \in S$ for $\texttt{Time}_{\texttt{bc}} + 4\Delta$ time.

- If A receives $(\texttt{Resend}, A, B, \texttt{rid}, m_i)$ signed by B from P_i, A records m_i.
- If A receives $(\texttt{Dispute}, P_i, B)$ with a valid signature from P_i, A adds $(\texttt{Dispute}, P_i, B)$ and the signature of P_i to `Proof` and adds P_i to D_B.
- Otherwise, A adds P_i to D_A.

If A records m_i from $t_a + 1$ distinct P_i, A reconstructs the secret m and broadcasts a confirmation $(\texttt{Confirm}, A, B, \texttt{rid})$. If $|D_B| \geq t_s + 1$, A broadcasts $(\texttt{Eliminate}, B, \texttt{Proof})$. Otherwise, A broadcasts $(\texttt{Eliminate}, A)$. Note that the last case happens only when A receives at most t_a shares and $|D_A| \leq t_s$, which implies that A recorded $(\texttt{Dispute}, A, P_i)$ for at least $n - t_a - t_s \geq t_s + 1$ distinct P_i.

We briefly argue that when the network is synchronous, no two honest parties will be set as disputed in the above process.

- For each pair of honest (P_i, B), when P_i enters the `rid`-th round and accepts $(\texttt{Resend}, A, B, \texttt{ReCount}, \texttt{rid}, S)$ broadcast by A, B is guaranteed to enter the `rid`-th round and accept $(\texttt{Resend}, A, B, \texttt{ReCount}, \texttt{rid}, S)$ broadcast by A within Δ time. This further ensures that P_i will receive $(\texttt{Resend}, A, B, \texttt{rid}, m_i)$ with a valid signature from B within 2Δ time. Thus, P_i will never send $(\texttt{Dispute}, P_i, B)$ to A.
- For each pair of honest (A, P_i), A will broadcast $(\texttt{Resend}, A, B, \texttt{ReCount}, \texttt{rid}, S)$ after A enters the `rid`-th round. Then P_i is guaranteed to enter the `rid`-th round and accepts $(\texttt{Resend}, A, B, \texttt{ReCount}, \texttt{rid}, S)$

from A within $\texttt{Time}_{\texttt{bc}} + \Delta$ time after A starts $\Pi_{\texttt{RBC}}$. By construction, after another 2Δ time, P_i will respond to A either with $(\texttt{Resend}, A, B, \texttt{rid}, m_i)$ signed by B or $(\texttt{Dispute}, P_i, B)$. In either case, A will get the response from P_i within $\texttt{Time}_{\texttt{bc}} + 4\Delta$ time and thus A will never record $(\texttt{Dispute}, A, P_i)$.

To preserve liveness, all parties will accuse A if he does not respond on time.

Protocol $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$

Advanced Termination

1: Recall that $\texttt{Time}_{\texttt{bc}}$ is the time required by $\Pi_{\texttt{RBC}}$ when the dealer is honest and the network is synchronous. All parties invoke $\Pi_{\texttt{BA}}$ and decide their input as follows: After entering the $\texttt{rid}$-th round, each party P_i waits for $3\Delta + \texttt{Time}_{\texttt{bc}} + n(4\Delta + \texttt{Time}_{\texttt{bc}})$ for the broadcast message from A.
   - If the received broadcast message is in the form of $(\texttt{Confirm}, A, B, \texttt{rid})$, $(\texttt{Eliminate}, B, \texttt{Proof})$ (with a valid $\texttt{Proof}$), or $(\texttt{Eliminate}, A)$, P_i sets his input of $\Pi_{\texttt{BA}}$ to be 1.
   - Otherwise P_i sets his input of $\Pi_{\texttt{BA}}$ to be 0.

2: All parties decide their output based on the result of $\Pi_{\texttt{BA}}$ and the final broadcast message from A.
   - If $\Pi_{\texttt{BA}}$ outputs 1 and the final broadcast message from A is $(\texttt{Confirm}, A, B, \texttt{rid})$, all parties output $(\texttt{Confirm}, A, B, \texttt{rid})$ and A outputs the received message m in addition.
   - If $\Pi_{\texttt{BA}}$ outputs 1 and the final broadcast message from A is $(\texttt{Eliminate}, B, \texttt{Proof})$ (with a valid $\texttt{Proof}$), all parties output $(\texttt{Eliminate}, B)$.
   - Otherwise, all parties output $(\texttt{Eliminate}, A)$.

The analysis of time limit $3\Delta + \texttt{Time}_{\texttt{bc}} + n(4\Delta + \texttt{Time}_{\texttt{bc}})$ is shown in Appendix D.2. When the network is synchronous, an honest P_i will never set his input of $\Pi_{\texttt{BA}}$ to be 0 for an honest A.

Remark 1. We can further bring down the overhead to constant if we assume a symmetric-key encryption scheme and A and B share a common secret key: Each message m sent from B to A is encrypted using the common secret key and only the cipher-text c is sent using $\Pi_{\texttt{P2P-Send}}$. In this way, there is no need to protect the secrecy of c even if A and B are both honest. Thus, in each resend attempt, A will only use one party as the intermediate to pass the cipher-text c rather than $t_a + 1$ parties to pass a degree-t_a Shamir sharing of the message m.

Realizing $\Pi_{\texttt{BC-Send}}$. The realization of $\Pi_{\texttt{BC-Send}}$ remains the same as described in Section 2.2. We omit the repetition and refer the readers to Section 2.2 for more details.

5.2 Realization of $\mathcal{F}_{\mathsf{Channel}}$

Now we are ready to present $\Pi_{\texttt{Channel}}$ that securely realizes $\mathcal{F}_{\mathsf{Channel}}$ in the network-agnostic setting.

Protocol $\Pi_{\texttt{Channel}}$

All parties receive as public parameters a set $\mathcal{P} = \{P_1, \ldots, P_n\}$, corruption threshold t_s, t_a, the delivery time-bound Δ for a message sent in the synchronous network, and a set $\mathcal{R}$ of valid round ids.

Initialization

1: All parties invoke $\mathcal{F}_{\mathsf{Setup}}$ to receive the signing keys and verification keys.
2: All parties set $R = |\mathcal{R}|$.

Online Phase

1: Each party P_i, upon receiving the request $(\texttt{rid}, m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ from the environment $\mathcal{E}$, checks that $\texttt{rid} \in R$. If true, P_i takes $m_0^{(i)}$ as the broadcast message, and $m_j^{(i)}$ as the P2P message for P_j. Then P_i invokes $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$ for every $A, B \in \mathcal{P}$, and invokes $\Pi_{\texttt{BC-Send}}(A, \texttt{rid})$ for every $A \in \mathcal{P}$.
2: All parties decide the set W as follows.
   - For each call of $\Pi_{\texttt{P2P-Send}}(P_i, P_j, \texttt{rid})$, if the output is $(\texttt{Eliminate}, P)$, all parties insert P in W.
   - For each call of $\Pi_{\texttt{BC-Send}}(P_i, \texttt{rid})$, if the output is $(\texttt{Eliminate}, P_i)$, all parties insert P_i in W.

3: For each party $P_i \in W$, P_i sets his output to be $(\texttt{rid}, W, \varnothing)$. For each party $P_i \notin W$, P_i sets his output to be $(\texttt{rid}, W, \{(m_0^{(j)}, m_i^{(j)})\}_{j \notin W})$.

Lemma 1. *Let n, t_s, t_a be positive integers such that $n = 2t_s + t_a + 1$. Let $(\texttt{Gen}, \texttt{Sign}, \texttt{Vrfy})$ be a signature scheme. Then $\Pi_{\texttt{Channel}}$ is a network-agnostic MPC protocol in the $\mathcal{F}_{\mathsf{Setup}}$-hybrid model that securely computes $\mathcal{F}_{\mathsf{Channel}}$ against an adversary who may control up to t_s corrupted parties when the network is synchronous, and t_a corrupted parties when the network is asynchronous.*

We refer the readers to Section E.1 for cost analysis and Section F.1 for the proof of Lemma 1.

Realization of $\mathcal{F}_{\mathsf{Channel}}$ with Computational Security and Constant Overhead. As we have mentioned in Remark 1, assuming a symmetric-key encryption scheme, if A and B share a common secret key, we can reduce the overhead of $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$ from $O(t_a)$ to $O(1)$. Now we show how to achieve $O(1)$ overhead for $\mathcal{F}_{\mathsf{Channel}}$ assuming PKI and symmetric-key encryption schemes. The key point is to establish a common random secret key between every pair of parties. In the following, the information-theoretic variant of $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$ refers to the version we presented above that uses degree-t_a Shamir secret sharings in the resend attempt (described in Section 2.2), and the computational variant refers to the version described in Remark 1 which assumes a symmetric-key encryption scheme and that A and B share a common secret key.

First we insert a new round id in $\mathcal{R}$, assuming it is $\texttt{rid} = 0$ for simplicity. In the initialization phase of $\Pi_{\texttt{Channel}}$, for each pair of parties (A, B), B samples a random secret key $\mathsf{sk}_{B \to A}$. Then all parties invoke the information-theoretic variant of $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid} = 0)$ for every $A, B \in \mathcal{P}$ to let B send $\mathsf{sk}_{B \to A}$ to A. Similar to the online phase, let W be the set that all parties decide. Then for every two $P_i, P_j \notin W$, P_i has received $\mathsf{sk}_{j \to i}$ from P_j. We set $\tilde{W} := W$. Now in the online phase, all parties only invoke $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$ and $\Pi_{\texttt{BC-Send}}(A, \texttt{rid})$ for $A, B \notin \tilde{W}$, and they always set $W := W \cup \tilde{W}$. That is, parties in $\tilde{W}$ are always regarded as eliminated parties. Intuitively, the security follows from the fact that in the synchronous setting, all parties in $\tilde{W}$ are guaranteed to be corrupted parties.

6 Agnostic MPC Compiler

In this section, we give the formal description of our compiler.

Required Functionalities in the Synchronous Setting. Let $f : (\{0,1\}^* \cup \{\perp\})^n \to \{0,1\}^*$ be the function that all parties wish to compute. Without loss of generality, we assume all parties receive the same output. Let $(\texttt{Share}, \texttt{Recon})$ be a (t_s, n)-robust secret sharing scheme with adaptive secrecy, online reconstruction, and efficient resampling.

We define the following two functionalities $\mathcal{F}_{\mathsf{Input}}, \mathcal{F}_{\mathsf{Comp}}$. Informally, $\mathcal{F}_{\mathsf{Input}}$ allows a single dealer D to distribute a robust secret sharing of his secret s. $\mathcal{F}_{\mathsf{Comp}}$ takes as input a set of robust secret sharings, computes the function f on the secrets of these robust secret sharings, and distributes the robust secret sharing of the function output to all parties.

Functionality $\mathcal{F}_{\mathsf{Input}}(D)$

$\mathcal{F}_{\mathsf{Input}}$ runs with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, a dealer $D \in \mathcal{P}$, and an adversary $\mathcal{S}$.

1: $\mathcal{F}_{\mathsf{Input}}$ receives the set $\mathcal{C}orr$ of up to t_s corrupted parties from $\mathcal{S}$.
2: $\mathcal{F}_{\mathsf{Input}}$ receives the input s from D.
3: $\mathcal{F}_{\mathsf{Input}}$ generates a random robust secret sharing of s, denoted by $[\![s]\!]$. $\mathcal{F}_{\mathsf{Input}}$ distributes the shares of $[\![s]\!]$ to all parties.
4: Whenever $\mathcal{F}_{\mathsf{Input}}$ receives $(\texttt{corrupt}, P_i)$ from $\mathcal{S}$, if $P_i \notin \mathcal{C}orr$ and $|\mathcal{C}orr| < t_s$, $\mathcal{F}_{\mathsf{Input}}$ adds P_i in $\mathcal{C}orr$ and sends P_i's share of $[\![s]\!]$ to P_i. If $P_i = D$, $\mathcal{F}_{\mathsf{Input}}$ also sends s to $\mathcal{S}$.

Functionality $\mathcal{F}_{\mathsf{Comp}}$

$\mathcal{F}_{\mathsf{Comp}}$ runs with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an adversary $\mathcal{S}$.

1: $\mathcal{F}_{\mathsf{Comp}}$ receives the set $\mathcal{C}orr$ of up to t_s corrupted parties from $\mathcal{S}$.
2: $\mathcal{F}_{\mathsf{Comp}}$ receives the set $\mathcal{I} \subset \mathcal{P}$ and for each $P_i \in \mathcal{I}$, $\mathcal{F}_{\mathsf{Comp}}$ receives the shares of $[\![x_i]\!]$ of honest parties.
3: For each $P_i \in \mathcal{I}$, $\mathcal{F}_{\mathsf{Comp}}$ reconstructs x_i from honest parties' shares of $[\![x_i]\!]$. For each $P_i \notin \mathcal{I}$, $\mathcal{F}_{\mathsf{Comp}}$ sets $x_i = \perp$. Then $\mathcal{F}_{\mathsf{Comp}}$ computes $y = f(x_1, \ldots, x_n)$ and generates n random robust secret sharings of y, denoted by $\{[\![y_i]\!]\}_{i=1}^n$. $\mathcal{F}_{\mathsf{Comp}}$ distributes the shares of $\{[\![y_i]\!]\}_{i=1}^n$ to all parties.
4: Whenever $\mathcal{F}_{\mathsf{Comp}}$ receives $(\texttt{corrupt}, P_i)$ from $\mathcal{S}$, if $P_i \notin \mathcal{C}orr$ and $|\mathcal{C}orr| < t_s$, $\mathcal{F}_{\mathsf{Comp}}$ adds P_i in $\mathcal{C}orr$ and sends P_i's shares of each $[\![s]\!] \in \{[\![x_i]\!]\}_{P_i \in \mathcal{I}} \cup \{[\![y_i]\!]\}_{i=1}^n$ to P_i.

6.1 Our Compiler

Now we are ready to introduce our compiler, which assumes (1) synchronous MPC protocols with GOD, $\Pi_{\mathtt{Input}}$ and $\Pi_{\mathtt{Comp}}$, that securely compute $\mathcal{F}_{\mathsf{Input}}$ and $\mathcal{F}_{\mathsf{Comp}}$ against t_s corrupted parties (among $2t_s + t_a + 1$ parties) with adaptive security assuming a broadcast channel, and (2) an asynchronous MPC protocol with GOD, $\Pi_{\mathtt{AMPC\text{-}GOD}}$, that securely compute $\mathcal{F}_{\mathsf{AMPC}}$ (defined in Section B.4) against t_a corrupted parties (among $t_s + 2t_a + 1$ parties). We further assume that the asynchronous MPC will take all honest parties' inputs when running in the synchronous network.

Protocol $\Pi_{\mathtt{Compiler}}$

Setup

Let $\Pi_{\mathtt{Input}}$ be a synchronous MPC with GOD that securely computes $\mathcal{F}_{\mathsf{Input}}$ against t_s corrupted parties with adaptive security assuming a broadcast channel. Suppose R_I is the (upper bound of the) number of rounds required by $\Pi_{\mathtt{Input}}$. Let $\Pi_{\mathtt{Comp}}$ be a synchronous MPC with GOD that securely computes $\mathcal{F}_{\mathsf{Comp}}$ against t_s corrupted parties with adaptive security assuming a broadcast channel. Suppose R_C is the (upper bound of the) number of rounds required by $\Pi_{\mathtt{Comp}}$. Let $\Pi_{\mathtt{AMPC\text{-}GOD}}$ be an asynchronous MPC protocol that securely computes $\mathcal{F}_{\mathsf{AMPC}}$ among $t_s + 2t_a + 1$ parties against t_a corruption.
All parties define $\mathcal{R} = \{(\Pi_{\mathtt{Input}}, P_i, j)\}_{i\in[n], j\in[R_I]} \cup \{(\Pi_{\mathtt{Comp}}, j)\}_{j\in[R_C]}$. Then all parties invoke $\mathcal{F}_{\mathsf{Channel}}$.

Input Phase

All parties initialize a set $\mathcal{I} = \varnothing$ at the beginning. For every $D \in \{P_1, \ldots, P_n\} \backslash \mathcal{W}$, from $\texttt{rid} = 1$ to R_I, all parties execute the following steps sequentially.

1: Each party $P_i \notin \mathcal{W}$ computes his $\texttt{rid}$-th round message $(m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ following the protocol $\Pi_{\mathtt{Input}}(D)$ where $m_0^{(i)}$ is the broadcast message, and $m_j^{(i)}$ is the P2P message to P_j.
2: Each party $P_i \in \mathcal{W}$ sets $(m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ to be all-0 strings.
3: Each party P_i invokes $\mathcal{F}_{\mathsf{Channel}}$ with input $((\Pi_{\mathtt{Input}}, D, \texttt{rid}), m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$.
4: Upon receiving the output from $\mathcal{F}_{\mathsf{Channel}}$, let W' denote the set received from $\mathcal{F}_{\mathsf{Channel}}$ for the $(\Pi_{\mathtt{Input}}, D, \texttt{rid})$-th round, each party P_i updates $\mathcal{W} := \mathcal{W} \cup W'$.
   – If $P_i \in \mathcal{W}$, P_i sets all his received messages to be all-0 strings in this round.
   – Otherwise, for each $P_j \notin \mathcal{W}$, P_i takes $(m_0^{(j)}, m_i^{(j)})$ as the messages received from P_j for all $P_j \notin \mathcal{W}$, and takes all-0 strings as the messages received from P_j for all $P_j \in \mathcal{W}$.
5: Depending on the size of $\mathcal{W}$ and $\texttt{rid}$, all parties run the following steps.
   – If $D \in \mathcal{W}$, all parties stop emulating $\Pi_{\mathtt{Input}}(D)$ and move on to the next dealer.
   – If $|\mathcal{W}| \leq t_s - t_a$,
     • If $\texttt{rid} = R_I$, P_i computes his output following $\Pi_{\mathtt{Input}}(D)$ and adds D in $\mathcal{I}$.
     • If $\texttt{rid} < R_I$, P_i moves to the $(\texttt{rid} + 1)$-th round of $\Pi_{\mathtt{Input}}(D)$.
   – Otherwise, all parties move to the fallback procedure.

Computation Phase

From $\texttt{rid} = 1$ to R_C, all parties execute the following steps.

1: Each party $P_i \notin \mathcal{W}$ computes his $\texttt{rid}$-th round message $(m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ following the protocol $\Pi_{\mathtt{Comp}}$ where $m_0^{(i)}$ is the broadcast message, and $m_j^{(i)}$ is the P2P message to P_j.

2: Each party $P_i \in \mathcal{W}$ sets $(m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ to be all-0 strings.
3: Each party P_i invokes $\mathcal{F}_{\mathsf{Channel}}$ with input $((\Pi_{\mathtt{Comp}}, \mathtt{rid}), m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$.
4: Upon receiving the output from $\mathcal{F}_{\mathsf{Channel}}$, let W' denote the set received from $\mathcal{F}_{\mathsf{Channel}}$ for the $(\Pi_{\mathtt{Comp}}, \mathtt{rid})$-th round, each party P_i updates $\mathcal{W} := \mathcal{W} \cup W'$.
- If $P_i \in \mathcal{W}$, P_i sets all his received messages to be all-0 strings in this round.
- Otherwise, for each $P_j \notin \mathcal{W}$, P_i takes $(m_0^{(j)}, m_i^{(j)})$ as the messages received from P_j for all $P_j \notin \mathcal{W}$, and takes all-0 strings as the messages received from P_j for all $P_j \in \mathcal{W}$.

5: Depending on the size of $\mathcal{W}$ and $\mathtt{rid}$, all parties run the following steps.
- If $|\mathcal{W}| \leq t_s - t_a$,
  - If $\mathtt{rid} = R_C$, P_i computes his output following $\Pi_{\mathtt{Comp}}$ and adds D in $\mathcal{I}$.
  - If $\mathtt{rid} < R_C$, P_i moves to the $(\mathtt{rid}+1)$-th round of $\Pi_{\mathtt{Comp}}$.
- Otherwise, all parties move to the fallback procedure.

Reconstruction Phase

For each $[\![y_i]\!]$, all parties send their shares to P_i. Then P_i tries to reconstruct the secret y as follows: Whenever P_i receives a share $\mathtt{sh}_j$ from a party P_j, P_i applies $\mathtt{Recon}$ on all shares he has received so far. If the output of $\mathtt{Recon}$ is not $\perp$, P_i takes the output of $\mathtt{Recon}$ as y and halts.

Fallback Procedure

Let $\mathcal{W}'$ be the set of the first $t_s - t_a$ parties in $\mathcal{W}$. Parties in $\mathcal{P} \backslash \mathcal{W}'$ invoke $\Pi_{\mathtt{AMPC\text{-}GOD}}$ to compute the output y. Then parties in $\mathcal{P} \backslash \mathcal{W}'$ send the output to parties in $\mathcal{W}'$ and halt. Each party in $\mathcal{W}'$, after receiving $t_s + t_a + 1$ values from parties in $\mathcal{P} \backslash \mathcal{W}'$, takes the majority as the output and halts.

Lemma 2. *Let n, t_s, t_a be positive integers such that $n = 2t_s + t_a + 1$. Let f be an n-ary function and $(\mathtt{Share}, \mathtt{Recon})$ be a (t_s, n)-robust secret sharing scheme with adaptive secrecy, online reconstruction, and efficient resampling. Suppose that*

- *$\Pi_{\mathtt{Input}}$ is a synchronous MPC with GOD that securely computes $\mathcal{F}_{\mathsf{Input}}$ against t_s corrupted parties with adaptive security assuming a broadcast channel;*
- *$\Pi_{\mathtt{Comp}}$ is a synchronous MPC with GOD that securely computes $\mathcal{F}_{\mathsf{Comp}}$ against t_s corrupted parties with adaptive security assuming a broadcast channel;*
- *$\Pi_{\mathtt{AMPC\text{-}GOD}}$ is an asynchronous MPC with GOD that securely computes $\mathcal{F}_{\mathsf{AMPC}}$ among $t_s + 2t_a + 1$ parties against t_a corrupted parties such that when running in the synchronous network, all honest parties' inputs will be considered.*

Then $\Pi_{\mathtt{Compiler}}$ is a network-agnostic MPC in the $\mathcal{F}_{\mathsf{Channel}}$-hybrid model with GOD that securely computes $\mathcal{F}_{\mathsf{sfe}}$ against an adversary who may control up to t_s corrupted parties when the network is synchronous, and t_a corrupted parties when the network is asynchronous.

We refer the readers to Section E.2 for cost analysis and Section F.2 for the proof of Lemma 2.

Final Results. Plugging the numbers in Figure 1, we obtain the following results.

Theorem 1. *Let n, t_s, t_a be positive integers such that $n = 2t_s + t_a + 1$ and κ be the security parameter. Let f be an n-ary function. Assuming a trusted setup for pseudo signatures, there is an unconditionally secure network-agnostic MPC protocol for f among n parties against a malicious adversary who controls up to t_s corrupted parties when the network is synchronous, and t_a corrupted parties when the network is asynchronous.*

For a function f with circuit size $|C|$, input size C_I, output size C_O, and circuit depth D, the MPC protocol requires setup size $O(t_a(n|C|\kappa + n^3 C_I \kappa + n^3 C_O \kappa) + n^6 D\kappa + n^9\kappa)$ and achieves communication complexity $O(n|C|\kappa + n^3 C_I \kappa + n^3 C_O \kappa + (n^3 + D)n^6 \kappa \log(nD) + n^8\kappa^2)$ in expectation in the optimistic case and $O(t_a(n|C|\kappa + n^3 C_I \kappa + n^3 C_O \kappa) + n^8\kappa^2 \cdot D + n^{14}\kappa^2)$ in expectation in the worst case.

Theorem 2. *Let n, t_s, t_a be positive integers such that $n = 2t_s + t_a + 1$ and κ be the security parameter. Let f be an n-ary function. Assuming PKI and symmetric-key encryption schemes, there is a computationally*

secure network-agnostic MPC protocol for f among n parties against a malicious adversary who controls up to t_s corrupted parties when the network is synchronous, and t_a corrupted parties when the network is asynchronous.

For a function f with circuit size $|C|$, input size C_I, output size C_O, and circuit depth D, the MPC protocol requires setup size $O(n^2\kappa)$ and achieves communication complexity $O(n|C|\kappa+n^3C_I\kappa+n^3C_O\kappa+(n^3+D)n^6\kappa\log(nD)+n^8\kappa^2)$ in expectation in the optimistic case and $O(n|C|\kappa+n^3C_I\kappa+n^3C_O\kappa+n^7\kappa^2\cdot D+n^{14}\kappa^2)$ in expectation in the worst case.

Acknowledgments. X. Ji and Y. Song were supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61033001, 61361136003. Y. Song was also supported in part by the Shanghai Qi Zhi Institute Innovation Program SQZ202313.

References

[AAPP24] Ittai Abraham, Gilad Asharov, Shravani Patil, and Arpita Patra. Perfect asynchronous MPC with linear communication overhead. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*, pages 280–309. Springer, Cham, May 2024.

[ABKL22] Andreea B. Alexandru, Erica Blum, Jonathan Katz, and Julian Loss. State machine replication under changing network conditions. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 681–710. Springer, Cham, December 2022.

[AC23] Ananya Appan and Ashish Choudhury. Network agnostic MPC with statistical security. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part II*, volume 14370 of *LNCS*, pages 63–93. Springer, Cham, November / December 2023.

[ACC22] Ananya Appan, Anirudh Chandramouli, and Ashish Choudhury. Perfectly-secure synchronous MPC with asynchronous fallback guarantees. In Alessia Milani and Philipp Woelfel, editors, *41st ACM PODC*, pages 92–102. ACM, July 2022.

[ACC23] Ananya Appan, Anirudh Chandramouli, and Ashish Choudhury. Network agnostic perfectly secure mpc against general adversaries. In *37th International Symposium on Distributed Computing (DISC 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

[AMN+20] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync HotStuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy*, pages 106–118. IEEE Computer Society Press, May 2020.

[BCLZL23] Renas Bacho, Daniel Collins, Chen-Da Liu-Zhang, and Julian Loss. Network-agnostic security comes (almost) for free in DKG and MPC. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 71–106. Springer, Cham, August 2023.

[BCV23] Nidhish Bhimrajka, Ashish Choudhury, and Supreeth Varadarajan. Network-agnostic perfectly secure message transmission revisited. In Anupam Chattopadhyay, Shivam Bhasin, Stjepan Picek, and Chester Rebeiro, editors, *INDOCRYPT 2023, Part II*, volume 14460 of *LNCS*, pages 25–44. Springer, Cham, December 2023.

[BCV24] Nidhish Bhimrajka, Ashish Choudhury, and Supreeth Varadarajan. Network-agnostic multi-party computation revisited (extended abstract). In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part II*, volume 14602 of *LNCS*, pages 171–204. Springer, Cham, April 2024.

[BDD+21] Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. TARDIS: A foundation of time-lock puzzles in UC. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 429–459. Springer, Cham, October 2021.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 169–188, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[BFO12] Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 663–680. Springer, Berlin, Heidelberg, August 2012.

[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.

[BJK+25a] Akhil Bandarupalli, Xiaoyu Ji, Aniket Kate, Chen-Da Liu-Zhang, Daniel Pöllmann, and Yifan Song. Velox: Scalable fair asynchronous mpc from lightweight cryptography. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pages 1799–1813, 2025.

[BJK+25b] Akhil Bandarupalli, Xiaoyu Ji, Aniket Kate, Chen-Da Liu-Zhang, and Yifan Song. Computationally efficient asynchronous MPC with linear communication and low additive overhead. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part IV*, volume 16003 of *LNCS*, pages 261–294. Springer, Cham, August 2025.

[BKL19] Erica Blum, Jonathan Katz, and Julian Loss. Synchronous consensus with optimal asynchronous fallback guarantees. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 131–150. Springer, Cham, December 2019.

[BKL21] Erica Blum, Jonathan Katz, and Julian Loss. Tardigrade: An atomic broadcast protocol for arbitrary network conditions. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 547–572. Springer, Cham, December 2021.

[BKR94] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In Jim Anderson and Sam Toueg, editors, *13th ACM PODC*, pages 183–192. ACM, August 1994.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *International conference on the theory and application of cryptology and information security*, pages 514–532. Springer, 2001.

[BSFO12] Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 663–680, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[BTHR07] Zuzana Beerliová-Trubíniová, Martin Hirt, and Micha Riser. Efficient byzantine agreement with faulty minority. In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 393–409. Springer, Berlin, Heidelberg, December 2007.

[BZL20] Erica Blum, Chen-Da Liu Zhang, and Julian Loss. Always have a backup plan: Fully secure synchronous MPC with asynchronous fallback. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 707–731. Springer, Cham, August 2020.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[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.

[CDD+99] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 311–326. Springer, Berlin, Heidelberg, May 1999.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 280–299. Springer, Berlin, Heidelberg, May 2001.

[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.

[CGHZ16] Sandro Coretti, Juan A. Garay, Martin Hirt, and Vassilis Zikas. Constant-round asynchronous multiparty computation based on one-way functions. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 998–1021. Springer, Berlin, Heidelberg, December 2016.

[CGWW24] Andrei Constantinescu, Diana Ghinea, Roger Wattenhofer, and Floris Westermann. Convex consensus with asynchronous fallback. In *38th International Symposium on Distributed Computing (DISC 2024)*, pages 15–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

[CHLZ21] Annick Chopard, Martin Hirt, and Chen-Da Liu-Zhang. On communication-efficient asynchronous MPC with adaptive security. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 35–65. Springer, Cham, November 2021.

[Coh16] Ran Cohen. Asynchronous secure multiparty computation in constant time. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 183–207. Springer, Berlin, Heidelberg, March 2016.

[CP15] Ashish Choudhury and Arpita Patra. Optimally resilient asynchronous mpc with linear communication complexity. In *Proceedings of the 16th International Conference on Distributed Computing and Networking*, pages 1–10, 2015.

[CP23] Ashish Choudhury and Arpita Patra. On the communication efficiency of statistically secure asynchronous mpc with optimal resilience. *Journal of Cryptology*, 36(2):13, 2023.

[DEP23] Ivan Damgård, Daniel Escudero, and Antigoni Polychroniadou. Phoenix: Secure computation in an unstable network with dropouts and comebacks. In Kai-Min Chung, editor, *ITC 2023*, volume 267 of *LIPIcs*, pages 7:1–7:21. Schloss Dagstuhl, June 2023.

[DHLZ21] Giovanni Deligios, Martin Hirt, and Chen-Da Liu-Zhang. Round-efficient byzantine agreement and multi-party computation with asynchronous fallback. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 623–653. Springer, Cham, November 2021.

[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.

[DLZ23] Giovanni Deligios and Chen-Da Liu-Zhang. Synchronous perfectly secure message transmission with optimal asynchronous fallback guarantees. In Foteini Baldimtsi and Christian Cachin, editors, *FC 2023, Part I*, volume 13950 of *LNCS*, pages 77–93. Springer, Cham, May 2023.

[DME24] Giovanni Deligios and Mose Mizrahi Erbes. Closing the efficiency gap between synchronous and network-agnostic consensus. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 432–461. Springer, 2024.

[DN03] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 247–264. Springer, Berlin, Heidelberg, August 2003.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[GLZS24] Vipul Goyal, Chen-Da Liu-Zhang, and Yifan Song. Towards achieving asynchronous MPC with linear communication and optimal resilience. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VIII*, volume 14927 of *LNCS*, pages 170–206. Springer, Cham, August 2024.

[GLZW22] Diana Ghinea, Chen-Da Liu-Zhang, and Roger Wattenhofer. Optimal synchronous approximate agreement with asynchronous fallback. In Alessia Milani and Philipp Woelfel, editors, *41st ACM PODC*, pages 70–80. ACM, July 2022.

[GLZW23] Diana Ghinea, Chen-Da Liu-Zhang, and Roger Wattenhofer. Multidimensional approximate agreement with asynchronous fallback. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 141–151, 2023.

[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, 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.

[GPS21] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Unconditional communication-efficient MPC via hall's marriage theorem. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 275–304, Virtual Event, August 2021. Springer, Cham.

[GPS22] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Sharing transformation and dishonest majority MPC with packed secret sharing. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 3–32. Springer, Cham, August 2022.

[GSZ20] Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed output delivery comes free in honest majority MPC. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 618–646. Springer, Cham, August 2020.

[HNP05] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Cryptographic asynchronous multiparty computation with optimal resilience (extended abstract). In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 322–340. Springer, Berlin, Heidelberg, May 2005.

[JLS24] Xiaoyu Ji, Junru Li, and Yifan Song. Linear-communication asynchronous complete secret sharing with optimal resilience. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VIII*, volume 14927 of *LNCS*, pages 418–453. Springer, Cham, August 2024.

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1:36–63, 2001.

[KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 477–498. Springer, Berlin, Heidelberg, March 2013.

[LZLM+19] Chen-Da Liu-Zhang, Julian Loss, Ueli Maurer, Tal Moran, and Daniel Tschudi. Robust MPC: Asynchronous responsiveness yet synchronous security. Cryptology ePrint Archive, Report 2019/159, 2019.

[MR21] Atsuki Momose and Ling Ren. Multi-threshold byzantine fault tolerance. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1686–1699. ACM Press, November 2021.

[PCR09] Arpita Patra, Ashish Choudhary, and C Pandu Rangan. Communication efficient statistical asynchronous multiparty computation with optimal resilience. In *International Conference on Information Security and Cryptology*, pages 179–197. Springer, 2009.

[PCR14] Arpita Patra, Ashish Choudhury, and C Pandu Rangan. Asynchronous byzantine agreement with optimal resilience. *Distributed computing*, 27(2):111–146, 2014.

[PP24] Shravani Patil and Arpita Patra. Perfectly-secure network-agnostic mpc with optimal resiliency. *Cryptology ePrint Archive*, 2024.

[PW96] Birgit Pfitzmann and Michael Waidner. *Information-theoretic pseudosignatures and byzantine agreement for $t \geq n/3$.* Citeseer, 1996.

[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.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology—CRYPTO'89 Proceedings 9*, pages 239–252. Springer, 1990.

[SHZI02] Junji Shikata, Goichiro Hanaoka, Yuliang Zheng, and Hideki Imai. Security notions for unconditionally secure signature schemes. In *Advances in Cryptology—EUROCRYPT 2002: International Conference on the Theory and Applications of Cryptographic Techniques Amsterdam, The Netherlands, April 28–May 2, 2002 Proceedings 21*, pages 434–449. Springer, 2002.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

A Ideal Functionality

Functionality $\mathcal{F}_{\mathsf{sfe}}$

$\mathcal{F}_{\mathsf{sfe}}$ is parameterized by a set $\mathcal{P} = \{P_1, \ldots, P_n\}$, an adversary $\mathcal{S}$, an n-party function $f : (\{0,1\}^* \cup \{\perp\})^n \to \{0,1\}^* \cup \{\perp\}$ and a flag $\mathsf{mode} \in \{\mathsf{synch}, \mathsf{asynch}\}$. Let t_s (resp. t_a) be the corruption in the synchronous (resp. asynchronous) mode.
$\mathcal{F}_{\mathsf{sfe}}$ has timeouts τ_0, τ_f for receiving the inputs and giving outputs of honest parties in the synchronous mode. For each party P_i, initialize an input value $x^{(i)} = \perp$ and output value $y^{(i)} = \perp$. Set $\mathsf{CoreSet} = \mathcal{H}$.

Initialization

1: $\mathcal{F}_{\mathsf{sfe}}$ receives the set $\mathcal{C}orr$ of corrupted parties from $\mathcal{S}$ and the network configuration $\mathsf{mode} \in \{\mathsf{synch}, \mathsf{asynch}\}$. $\mathcal{F}_{\mathsf{sfe}}$ checks that $|\mathcal{C}orr| \leq t_s$ if $\mathsf{mode} = \mathsf{synch}$ and $|\mathcal{C}orr| \leq t_a$ if $\mathsf{mode} = \mathsf{asynch}$. If not, $\mathcal{F}_{\mathsf{sfe}}$ gives up the security[a].

Synchronous: $\mathsf{mode} = \mathsf{synch}$

1: Upon receiving an input (Input, v) from $P_i \in \mathcal{P}$ by time τ_0, set $x^{(i)} = v$. Send a message (Input, P_i) to $\mathcal{S}$.
2: At time τ_0, if $x^{(i)}$ has been set to a value different from $\perp$ for every honest $P_i \in \mathcal{P}$, then compute $y = f(x^{(1)}, \ldots, x^{(n)})$ and set $y^{(i)} = y$ for every $P_i \in \mathcal{P}$. Otherwise, give up the security.
3: Upon receiving an input $(\mathsf{Deliver}, P_i)$ from $\mathcal{S}$ for an honest party P_i, if the output has been computed but not yet delivered to P_i, output $y^{(i)}$ to P_i.
4: Upon reaching time τ_f, output $y^{(i)}$ to every honest P_i that has not been delivered its output.

Asynchronous: $\mathsf{mode} = \mathsf{asynch}$

1: Upon receiving an input (Input, v) from $P_i \in \mathcal{P}$, set $x^{(i)} = v$. Send a message (Input, P_i) to $\mathcal{S}$.
2: Upon receiving an input $\mathsf{CoreSet}'$ from $\mathcal{S}$ for the first time, verify that $\mathsf{CoreSet}'$ is a subset of $\mathcal{P}$ of size at least $n - t_s$, else ignore the message. Then record $\mathsf{CoreSet} = \mathsf{CoreSet}'$, and for every $P_i \notin \mathsf{CoreSet}$, set $x^{(i)} = \perp$.
3: If $x^{(i)}$ has been set to a value different from $\perp$ for every honest $P_i \in \mathsf{CoreSet}$, then compute $y = f(x^{(1)}, \ldots, x^{(n)})$ and generate a request-based delayed output $y^{(i)} = y$ for every $P_i \in \mathcal{P}$.

[a] By giving up the security, $\mathcal{F}_{\mathsf{sfe}}$ sends its view to $\mathcal{S}$ and allows $\mathcal{S}$ to arbitrarily decide the output of honest parties and the delivery time.

B Additional Preliminary

B.1 Pseudo Signature Scheme

We recall the definition of the pseudo signature scheme and the construction in [SHZI02] as follows.

Definition 5. *[PW96, SHZI02] A $\mathcal{P}$-verifiable signature scheme $\Pi = (\mathtt{Gen}, \mathtt{Sign}, \mathtt{Vrfy})$ is said to be (ℓ_s, ℓ_v)-information-theoretic security if it satisfies the following properties: Let $\mathcal{O}_{\mathtt{Sign}}$ denote a signing oracle (on input m, $\mathcal{O}_{\mathtt{Sign}}$ invokes $\mathtt{Gen}$ and outputs $\sigma = \mathtt{Sign}(m, \mathsf{sk})$) and $\mathcal{O}_{\mathtt{Vrfy}}$ denote a verification oracle (on input (m, σ, i), $\mathcal{O}_{\mathtt{Vrfy}}$ outputs $\mathtt{Vrfy}(m, \sigma, \mathsf{vk}_i)$). Also, let $\mathcal{A}$ denote an adversary that makes at most ℓ_s and ℓ_v calls to $\mathcal{O}_{\mathtt{Sign}}$ and $\mathcal{O}_{\mathtt{Vrfy}}$, respectively, and gets to see the verification keys indexed by some set $\mathcal{I} \subsetneq \mathcal{P}$.*

– ***Completeness.*** *A valid signature will be accepted:*

$$\Pr\left[\begin{array}{c}\mathtt{Gen}(1^\kappa, \mathcal{P}, \ell_s, \ell_v) \to (\mathsf{sk}, \mathbf{vk}) : \\ \mathtt{Sign}(m, \mathsf{sk}) \to \sigma; \forall i \in \mathcal{P}, \mathtt{Vrfy}(m, \sigma, \mathsf{vk}_i) = 1\end{array}\right] = 1$$

– ***Unforgeability.*** *A computationally unbounded adversary cannot (except with negligible probability) generate a valid signature on a message of his choice, other than the one he queries* $\mathcal{O}_{\texttt{Sign}}$*. Formally,*

$$\Pr\left[\begin{array}{c}\texttt{Gen}(1^\kappa,\mathcal{P},\ell_s,\ell_v)\to(\mathsf{sk},\mathbf{vk});\ \textit{for some}\ \mathcal{I}\subsetneq\mathcal{P}:\\ \mathcal{A}(\{\mathsf{vk}_i\}_{i\in\mathcal{I}})\to(m,\sigma), m\ \textit{was not queried to}\ \mathcal{O}_{\texttt{Sign}},\\ \exists j\in\mathcal{P}\setminus\mathcal{I}\quad s.t.\quad \texttt{Vrfy}(m,\sigma,\mathsf{vk}_j)=1\end{array}\right]=negl(\kappa).$$

– ***Transferability.*** *A computationally unbounded adversary cannot (except with negligible probability) create a signature that is accepted by some parties and rejected by some other parties even after seeing* ℓ_s *valid signatures and verifying* ℓ_v *signatures. Formally,*

$$\Pr\left[\begin{array}{c}\texttt{Gen}(1^\kappa,\mathcal{P},\ell_s,\ell_v)\to(\mathsf{sk},\mathbf{vk}),\ \textit{for some}\ \mathcal{I}\subsetneq\mathcal{P}:\\ \mathcal{A}(\mathsf{sk},\{\mathsf{vk}_i\}_{i\in\mathcal{I}})\to(m,\sigma),\\ \exists i,j\in\mathcal{P}\setminus\mathcal{I}\quad s.t.\quad \texttt{Vrfy}(m,\sigma,\mathsf{vk}_i)\neq\texttt{Vrfy}(m,\sigma,\mathsf{vk}_j)\end{array}\right]=negl(\kappa).$$

In their construction, they do not limit the upper bound of the verification times, and the input parameter ℓ_v for $\texttt{Gen}$ is omitted.

Algorithm Pseudo Signature

Key Generation: $\texttt{Gen}(1^\kappa,\mathcal{P},\ell_s)\to(\mathsf{sk},\mathbf{vk})$

1: Assuming that $|\mathcal{P}|=n$, for $(j,k)\in\{0,\ldots,n-1\}\times\{0,\ldots,\ell_s\}$, randomly sample $a_{j,k}\in\mathbb{F}$ and set the signing key to be the following multi-variate polynomial

$$\mathsf{sk}:=f(y_1,\ldots,y_{n-1},x)=\sum_{k=0}^{\ell_s}a_{0,k}x^k+\sum_{j=1}^{n-1}\sum_{k=0}^{\ell_s}a_{j,k}y_jx^k$$

2: For each $i\in[n]$, randomly sample vector $\boldsymbol{v}_i=(v_{i,1},\ldots,v_{i,n-1})\in\mathbb{F}^{n-1}$ and set the ith verification key to be

$$\mathsf{vk}_i=(\boldsymbol{v}_i,f_{\boldsymbol{v}_i}(x)),$$

where $f_{\boldsymbol{v}_i}(x)=f(v_{i,1},\ldots,v_{i,n-1},x)$.

Signature Generation: $\texttt{Sign}(m,\mathsf{sk})\to\sigma$

1: The algorithm for signing a message $m\in\mathbb{F}$ given the above signing key is the following polynomial

$$\texttt{Sign}(m,\mathsf{sk}):=g(y_1,\ldots,y_{n-1}):=f(y_1,\ldots,y_{n-1},m)$$

Signature Verification: $\texttt{Vrfy}(m,\sigma,\mathsf{vk}_i)\to b$

1: The algorithm for verifying a signature $\sigma=g(y_1,\ldots,y_{n-1})$ on a given message m using the ith verification key is

$$\texttt{Vrfy}(m,\sigma,\mathsf{vk}_i)=\begin{cases}1\text{ if }g(\boldsymbol{v_i})=f_{\boldsymbol{v}_i}(m)\\ 0\text{ otherwise}\end{cases}$$

For the above construction of information-theoretic security $\mathcal{P}$-verifiable signature scheme in [SHZI02], it requires that $|\mathbb{F}|=\Omega(2^\kappa)$ and the sizes of the signing key and each verification key are $\mathcal{O}(n\ell_s)$, and $\mathcal{O}(\ell_s+n)$ field elements, respectively. The size of a signature on a message in $\mathbb{F}$ is $\mathcal{O}(n)$ field elements. Thus to sign N messages of size M bits in total (which can be viewed as signing $\ell_s\leq M/\kappa+N$ field elements),

– The size of the signing key is bounded by $\mathcal{O}(Mn+Nn\kappa)$ bits.
– The size of each verification key is bounded by $\mathcal{O}(M+(N+n)\kappa)$ bits.
– The total signature size is bounded by $\mathcal{O}(Mn+Nn\kappa)$ bits.

Theorem 3. *([SHZI02]). Assuming* $\mathbb{F}=\Omega(2^\kappa)$ *and* $\ell_s=poly(\kappa)$*, the above signature scheme is an information-theoretically* $(\ell_s,poly(\kappa))$*-secure* $\mathcal{P}$*-verifiable signature scheme.*

B.2 Ensure Termination in Synchronous Network with Bounded Time Difference

By similar time, we mean that when an honest party terminates, all honest parties are guaranteed to terminate within Δ time. Note that this property is not guaranteed by the t-Termination property unless all parties participate in the BA protocol at the same time. To achieve this goal, a naive solution is as follows.

1. All parties invoke a network-agnostic BA protocol to agree on a common output m. When a party terminates BA, he forwards m to all parties along with his signature.
2. We define an *early cut-off proof* by a message m with t_s+1 distinct parties' valid signatures on m. When a party gets or receives a valid *early cut-off proof*, he sets his output of BA as the message m in this proof, then forwards this proof to all parties and terminates.

The above construction ensures that all honest parties will terminate within at most Δ time difference in the synchronous network. Let σ denote the signature size of an $|m|$-bit message. We use the construction in [DME24] to realize the network-agnostic BA. For their construction, it relies on the building blocks of a synchronous BA (SBA) protocol, an asynchronous BA (ABA) protocol, and a signature scheme. In our setting, since we do not assume threshold signature as in [DME24], we will replace their choices of SBA, ABA by the constructions in [BTHR07], [PCR14] respectively. In the following, we use κ to denote the security parameter.

For SBA constructed based on [BTHR07], it is information-theoretically secure and built based on a pseudo-signature [SHZI02]. For the SBA in [BTHR07], the round complexity is $\mathcal{O}(n)$. For communication complexity, each party needs to sign $\mathcal{O}(1)$ messages of size $\mathcal{O}(|m|)$. The message complexity is $\mathcal{O}(n^2)$, and the message size between each pair of parties is $\mathcal{O}(|m|+|\sigma|n)$, thus in total $\mathcal{O}(|m|n^2+|\sigma|n^3)$ bits per $|m|$ bits agreement.

For ABA in [PCR14], it guarantees termination with $1-\epsilon$ probability. Let κ be the security parameter, if we set $\epsilon = 1/2^\kappa$, with overwhelming probability, the expected round complexity is $\mathcal{O}(1)$ and the expected communication complexity is $\mathcal{O}(|m|n^4\kappa)$ bits per $|m|$ bits agreement. In the worst case, it will terminate with overwhelming probability $1-1/2^\kappa$ within $\mathcal{O}(\kappa)$ rounds.

For the construction in [DME24], excluding SBA, ABA, each party needs to sign $O(1)$ messages of size $O(|m|)$. The message complexity is $\mathcal{O}(n^2)$, and the message size between each pair of parties is $\mathcal{O}(|m|+|\sigma|n)$ bits, thus in total $\mathcal{O}(|m|n^2+|\sigma|n^3)$ bits. For the *early cut-off* procedure, the total communication cost is $\mathcal{O}(|m|n^2+|\sigma|n^3)$ bits.

BA with Information-Theoretic Security. To achieve an unconditionally secure BA, we use the pseudo signature scheme in [SHZI02]. The cost is summarized as follows.

- The expected-case round complexity is $\mathcal{O}(n)$. The worst-case round complexity is $\mathcal{O}(n+\kappa)$.
- For each party P_i, P_i holds a signing key of size bounded by $\mathcal{O}(|m|n+n\kappa)$ bits, and every party P_j holds a verification key of size bounded by $\mathcal{O}(|m|+n\kappa)$ bits.
- The expected-case communication is $\mathcal{O}(|m|n^4\kappa)$ bits.

BA with Computational Security. When using a computationally secure signature scheme, the cost is summarized as follows.

- The expected-case round complexity is $\mathcal{O}(n)$. The worst-case round complexity is $\mathcal{O}(n+\kappa)$.
- For each party P_i, P_i holds a signing key of size $\mathcal{O}(\kappa)$ bits, and every party P_j holds a verification key of size $\mathcal{O}(\kappa)$ bits.
- The expected-case communication is $\mathcal{O}(|m|n^4\kappa)$ bits.

B.3 Definition of Robust Secret Sharing

We give the formal definition of robust secret sharing below.

Definition 6 (Robust Secret Sharing). *Let t, n be positive integers and κ be the security parameter. Let S be the secret space. A (t, n)-robust secret sharing scheme contains two algorithms* $(\mathtt{Share}, \mathtt{Recon})$ *with the following syntax:*

- $\mathtt{Share}(\cdot)$ *takes as input a secret $s \in S$ and generates a vector of n strings $(\mathtt{sh}_1, \ldots, \mathtt{sh}_n)$, denoted by $[\![s]\!]$. The i-th entry $\mathtt{sh}_i$ is regarded as the i-th share.*
- $\mathtt{Recon}(\cdot)$ *takes as input a vector of n strings $(\mathtt{sh}_1, \ldots, \mathtt{sh}_n)$ and outputs a value s.*

It satisfies the following two requirements:

- *Secrecy: For all $s, s' \in S$ and for all subset $\mathcal{C}orr$ of at most t identities, the statistical distance of the following two distributions is negligible in κ.*

$$(\mathtt{sh}_1, \ldots, \mathtt{sh}_n) \leftarrow \mathtt{Share}(s) : (\mathtt{sh}_i)_{i \in \mathcal{C}orr} \approx_s (\mathtt{sh}'_1, \ldots, \mathtt{sh}'_n) \leftarrow \mathtt{Share}(s') : (\mathtt{sh}'_i)_{i \in \mathcal{C}orr}$$

- *Robust Reconstruction: For all $s \in S$, for all subset $\mathcal{C}orr$ of at most t identities, and for all adversary $\mathcal{A}$,*

$$\Pr\left[\begin{array}{c}(\mathtt{sh}_1, \ldots, \mathtt{sh}_n) \leftarrow \mathtt{Share}(s), (\mathtt{sh}'_i)_{i \in \mathcal{C}orr} \leftarrow \mathcal{A}((\mathtt{sh}_i)_{i \in \mathcal{C}orr}) : \\ \mathtt{Recon}(\{(\mathtt{sh}_i)_{i \notin \mathcal{C}orr}, (\mathtt{sh}'_i)_{i \in \mathcal{C}orr}\}) = s\end{array}\right] = 1 - \mathit{negl}(\kappa).$$

where $\mathit{negl}(\kappa)$ denotes a negligible function in κ.

In this work, we require the following three additional properties for the robust secret sharing scheme to build our construction, and we give an instantiation of such a robust secret sharing sheme in Section C.

Definition 7 (Adaptive Secrecy). *We say a (t, n)-robust secret sharing scheme achieves* adaptive *secrecy if any adversary $\mathcal{A}$ cannot win the following game with non-negligible advantage over $1/2$:*

1. *$\mathcal{A}$ chooses two secrets $s_0, s_1 \in S$ and sends them to the challenger.*
2. *The challenger samples a random bit b and generates $(\mathtt{sh}_1, \ldots, \mathtt{sh}_n) \leftarrow \mathtt{Share}(s_b)$.*
3. *$\mathcal{A}$ and the challenger interacts t rounds. In each round, $\mathcal{A}$ chooses an index $i \in \{1, \ldots, n\}$ and sends it to the challenger. Then the challenger replies $\mathtt{sh}_i$ to $\mathcal{A}$.*
4. *Finally $\mathcal{A}$ outputs a bit b'. If $b' = b$, $\mathcal{A}$ wins. Otherwise, $\mathcal{A}$ loses.*

Definition 8 (Online Reconstruction). *We say a (t, n)-robust secret sharing scheme is online reconstructable if for all $s \in S$, for all subset $\mathcal{C}orr$ of at most t identities, and for all adversary $\mathcal{A}$,*

$$\Pr\left[\begin{array}{c}(\mathtt{sh}_1, \ldots, \mathtt{sh}_n) \leftarrow \mathtt{Share}(s), ((\mathtt{sh}'_i)_{i \in \mathcal{C}orr}, I) \leftarrow \mathcal{A}((\mathtt{sh}_i)_{i \in \mathcal{C}orr}) : \\ \mathtt{Recon}(\{(\mathtt{sh}_i)_{i \in I \setminus \mathcal{C}orr}, (\mathtt{sh}'_i)_{i \in I \cap \mathcal{C}orr}\}) \in \{\perp, s\}\end{array}\right] = 1 - \mathit{negl}(\kappa).$$

where $\mathit{negl}(\kappa)$ denotes a negligible function in κ.

Definition 9 (Efficient Resampling). *We say a (t, n)-robust secret sharing scheme achieves efficient resampling if there exists a P.P.T. algorithm* $\mathtt{Sample}$ *such that for all $s \in S$ and for all subset $\mathcal{C}orr$ of at most t identities, the following two distributions are statistically indistinguishable.*

$$\begin{aligned}&(\mathtt{sh}_i)_{i \in [n]} \leftarrow \mathtt{Share}(s) : (\mathtt{sh}_i)_{i \in \mathcal{C}orr}, (\mathtt{sh}'_i)_{i \notin \mathcal{C}orr} \\ \approx_s\ &(\mathtt{sh}_i)_{i \in [n]} \leftarrow \mathtt{Share}(s), (\mathtt{sh}'_i)_{i \notin \mathcal{C}orr} \leftarrow \mathtt{Sample}((\mathtt{sh}_i)_{i \in \mathcal{C}orr}, s) : (\mathtt{sh}_i)_{i \in \mathcal{C}orr}, (\mathtt{sh}'_i)_{i \notin \mathcal{C}orr}.\end{aligned} \tag{1}$$

Remark 2. With the property of online reconstruction, a robust secret sharing $[\![s]\!]$ can be reconstructed to the receiver R in the asynchronous setting as follows: Each time R receives a new share from some party, R applies the reconstruction algorithm on all shares he has received. If the reconstruction result is not $\perp$, R takes the result as the secret and halts.

According to Definition 8, the reconstruction algorithm will either output the correct secret or a failure symbol $\perp$. This ensures that with overwhelming probability, R does not output an incorrect result. On the other hand, since honest parties will eventually send their shares to R, with shares from all honest parties, the reconstruction algorithm will eventually succeed guaranteed by Definition 6.

B.4 Functionality of Asynchronous MPC

For our compiler, the required functionality $\mathcal{F}_{\mathsf{AMPC}}$ for the fall back procedure is defined below.

Functionality $\mathcal{F}_{\mathsf{AMPC}}$

$\mathcal{F}_{\mathsf{AMPC}}$ runs with parties $\mathcal{P}' \subset \{P_1, \ldots, P_n\}$ of size $n' = t_s + 2t_a + 1$ and an adversary $\mathcal{S}$. It is parameterized with a function $f : (\{0,1\}^* \cup \{\perp\})^n \to \{0,1\}^*$. Each party P_i has an initial input value $x_i = \perp$ and output value $y = \perp$.

1: Upon receiving an input v from a party $P_i \in \mathcal{P}'$, set $x_i = v$ and notify $\mathcal{S}$.
2: Upon receiving the core-set input $S \subset \mathcal{P}'$ of size at least $n' - t_a$ from the adversary $\mathcal{S}$ for the first time, record it and for every party $P_i \notin S$, set its input to $x_i = \perp$.
3: Upon receiving inputs from all parties in the core-set, $\mathcal{F}_{\mathsf{AMPC}}$ evaluates the function f on the given inputs and obtains output $y = f(x_1, \ldots, x_n)$.
4: $\mathcal{F}_{\mathsf{AMPC}}$ generates a request-based delayed output to send $P_i \in \mathcal{P}'$ the output y.

C Instantiation of the Robust Secret Sharing Scheme

The sharing algorithm $\mathtt{Share}(s)$ first samples a degree-t Shamir secret sharing, denoted by $(s_1, \ldots, s_n)$. Then for every i, the sharing algorithm samples a random value $\kappa^{(i)}$, and for every pair (i, j), the sharing algorithm samples a pair of values $a_j^{(i)}, b_i^{(j)}$ subject to $a_j^{(i)} + b_i^{(j)} = \kappa^{(i)} \cdot s_j$. Finally, the sharing algorithm sets the i-th share $\mathtt{sh}_i$ to be $(s_i, \kappa^{(i)}, \{a_j^{(i)}, b_j^{(i)}\}_{j=1}^n)$.

The reconstruction algorithm takes as input $(\mathtt{sh}_1, \ldots, \mathtt{sh}_n)$. Then for every pair (i, j), sets $s_j^{(i)} = s_j$ if $a_j^{(i)} + b_i^{(j)} = \kappa^{(i)} \cdot s_j$ and $s_j^{(i)} = \perp$ otherwise. Next, for all $i \in \{1, \ldots, n\}$, if at least $t + 1$ values of $\{s_j^{(i)}\}_{j=1}^n$ are not $\perp$, and all non-empty values lie on a valid degree-t polynomial, set $s^{(i)}$ to be the evaluation result at point 0. Otherwise, set $s^{(i)} = \perp$. If at least $t + 1$ values of $\{s^{(i)}\}_{i=1}^n$ are identical, say they are equal to s, set the reconstruction result as s. Otherwise, output $\perp$.

We will now prove that the above instantiation of an RSS scheme satisfies *Secrecy*, *Robust Reconstruction*, *Adaptive Secrecy*, *Online Reconstruction* and *Efficient Resampling*.

Adaptive Secrecy. To prove *Adaptive Secrecy*, we consider two experiments: $\mathtt{Exp}_{RSS}$, where the sharing is computed for secret s according to our previously defined RSS scheme and $\mathtt{Exp}_{adapt.}$ where an adversary is allowed to adaptively choose t shares, which will be sampled uniformly at random. Only after these shares are fixed are the remaining $n - t$ shares computed such that the reconstruction algorithm outputs s.

Experiment $\mathtt{Exp}_{RSS}$

Input: s
1: $(\mathtt{sh}_1, \ldots, \mathtt{sh}_n) \leftarrow \mathtt{Share}(s)$
2: **return** $\{(s_i, \kappa^{(i)}, \{a_j^{(i)}, b_j^{(i)}\}_{j=1}^n)\}_{i=1}^n$

Experiment $\mathtt{Exp}_{adapt.}$

Input: s
1: $\mathcal{C}orr = \varnothing$
2: **while** $|\mathcal{C}orr| < t$ **do**
3: $\quad corr \leftarrow \mathcal{A}(\mathcal{P} \setminus \mathcal{C}orr)$
4: $\quad s_{corr} \xleftarrow{\$} \mathbb{F}$; $\kappa^{(corr)} \xleftarrow{\$} \mathbb{F}$
5: $\quad$ **for** $j \in \mathcal{C}orr$ **do**
6: $\quad\quad a_j^{corr} = s_j \cdot \kappa^{(corr)} - b_{corr}^{(j)}$
7: $\quad\quad b_j^{corr} = s_j \cdot \kappa^{(corr)} - a_{corr}^{(j)}$
8: $\quad$ **end for**
9: $\quad$ **for** $j \in \mathcal{P} \setminus \mathcal{C}orr$ **do**
10: $\quad\quad a_j^{corr} \xleftarrow{\$} \mathbb{F}$

```
        b_j^{corr} ←$ 𝔽
    end for
    Corr ← Corr ∪ {corr}
    give {(s_corr, κ^{(corr)}, {a_j^{(corr)}, b_j^{(corr)}}_{j∈P})} to A
end while
// All values in {(s_i, κ^{(i)}, {a_j^{(i)}, b_j^{(i)}}_{j∈P})}_{i∈Corr} are random and the secret s has not been used to create these shares.
Use the first i ∈ P \ Corr to choose s_i such that points (s_j)_{j∈Corr} and s_i together define a polynomial f with f(0) = s.
interpolate (s_i)_{i∈P\Corr} using f
for i ∈ P \ Corr do
    κ^{(i)} ←$ 𝔽
    for j ∈ Corr do
        a_j^{(i)} = κ^{(i)} · s_j − b_i^{(j)}  // b_i^{(j)} is part of an adversary's share
    end for
    for j ∈ P \ Corr do
        if b_i^{(j)} is not defined then
            b_i^{(j)} ← 𝔽
        end if
        a_j^{(i)} = κ^{(i)} · s_j − b_i^{(j)}
    end for
end for
return {(s_i, κ^{(i)}, {a_j^{(i)}, b_j^{(i)}}_{j∈P}^n)}_{i∈P}^n
```

The shares that the adversary chose are sampled uniformly at random, while shares for honest parties are computed such that the reconstruction outputs the correct secret.

We invoke $(\mathtt{sh}_i, \ldots, \mathtt{sh}_n) \leftarrow \mathtt{Exp}_{RSS}(s)$ and $(\mathtt{sh}'_1, \ldots, \mathtt{sh}'_n) \leftarrow \mathtt{Exp}_{adapt.}(s)$. All shares in $(\mathtt{sh}'_{\mathcal{C}orr})$ cannot possibly contain any information on s, as these values were chosen uniformly at random without using s. Note that the distributions for the first t shares are statistically close, proving that $(\mathtt{sh}'_{\mathcal{C}orr})$ is independent of the secret.

Secrecy. *Secrecy* follows directly from *Adaptive Secrecy*. Instead of letting the adversary choose $\mathcal{C}orr$ adaptively, it has to commit to $\mathcal{C}orr$ beforehand. This (static) adversary has fewer capabilities than the adaptive adversary. It follows that the scheme satisfies secrecy.

Robust Reconstruction. The argument for *Robust Reconstruction* is similar to the proof for *Online Reconstruction*, with the difference being that the reconstruction has access to all n shares (t of them are coming from potentially corrupted parties) and $\perp$ is no longer allowed as output.

Intuitively, after a dealer distributed shares for a secret s, corrupted parties should be unable to change the reconstruction result to $s' \neq s$, except with negligible probability. For the reconstruction to output s', the adversary has to change at least $t+1$ of the values $\{s^{(i)}\}_{i=1}^n$. The adversary can control the values $(s_j^{(i)})_{j,i\in\mathcal{C}orr}$ and force $\perp$ for $(s_j^{(i)})_{j\in\mathcal{P}\setminus\mathcal{C}orr, i\in\mathcal{C}orr}$. The values $(s_j^{(i)})_{j,i\in\mathcal{P}\setminus\mathcal{C}orr}$ come from honest parties and cannot be changed or dropped by the adversary, except with negligible probability. As $|\mathcal{C}orr| \leq t$, for each $j \in \mathcal{P}\setminus\mathcal{C}orr$, there are at least $t+1$ non-empty $s_j^{(i)}$ from honest parties. Therefore, at least $t+1$ polynomial interpolations lead to the identical secret $s^{(i)} = s$, which is then accepted as the output.

Online Reconstruction. Let us assume that the reconstruction algorithm outputs $s' \notin \{s, \perp\}$. This implies that at least $t+1$ values of $(s^{(i)})_{i\in I\cup\mathcal{C}orr}$ must have been s', which is only possible if $|I \cup \mathcal{C}orr| \geq t+1$. Assume the best case for the adversary[6] where $|\mathcal{C}orr| = t$ and $|I| = 1$. For $t+1$ values in $(s^{(i)})_{i\in I\cup\mathcal{C}orr}$ to be equal to s', each $(s_j^{(i)})_{j=1}^n$ must have contained at least $t+1$ non-empty values that all lie on a valid degree-t polynomial. Note that only for pairs $(i,j) \in \mathcal{C}orr \times \mathcal{C}orr$, the adversary can control the value $s_j^{(i)}$.

[6] When I becomes larger, more honest parties participate, which makes it harder for the adversary.

Specifically, for any $j \in I \cup \mathcal{C}orr$, the adversary controls the values in $(s_j^{(i)})_{i \in \mathcal{C}orr}$. Assume there are no $\perp$ values in $(s_j^{(i)})_{i \in I \cup \mathcal{C}orr}$ and the values define a valid degree-t polynomial with secret $s' \neq s$ at position 0. $(s_j^{(i)})_{i \in I}$ is outside the direct control of the adversary—at most the adversary can force values in $(s_j^{(i)})_{i \in I}$ to be $\perp$, in which case $s^{(i)} = \perp$. If the values in this set are all non-empty, then they define a valid degree-t polynomial for which the secret is s and $s^{(i)} = s$. If $(s_j^{(i)})_{i \in I}$ contains $\perp$, then $s^{(i)} = \perp$. In either case, there will not be $t+1$ identical values $s' \neq s$ in $(s^{(i)})_{i \in I \cup \mathcal{C}orr}$ such that the reconstruction would accept $s' \neq \perp$ and must therefore output $\perp$. Note that with each additional party in I, more values in $(s^{(i)})_{i \in I \cup \mathcal{C}orr}$ will be set to s or $\perp$. For any value $s' \neq s$, the adversary has satisfy $a_j^{(i)} + b_i^j = \kappa^{(i)} \cdot s_j$, which it can only do with negligible probability when P_i or P_j is honest. Once the threshold of $t+1$ identical values in $(s^{(i)})_{i \in I \cup \mathcal{C}orr}$ is reached, the reconstruction will output s.

Efficient Resampling. Intuitively, this property ensures that honest parties' shares can be replaced with new shares, given only the adversary's shares and the secret s, such that all shares together still define the same secret s. The shares $(s_i)_{i \notin \mathcal{C}orr}$ of honest parties can be interpolated with the polynomial of degree t that goes through the adversary's Shamir shares $(s_i)_{i \in \mathcal{C}orr}$ and s at position 0. To create the RSS shares $(\mathtt{sh}_i)_{i \notin \mathcal{C}orr}$, the values $(\kappa^{(i)}, \{a_j^{(i)}, b_j^{(i)}\}_{j \notin \mathcal{C}orr})$ need to be chosen such that they are compatible with the existing shares of the adversary. For this, we refer to $\mathtt{Exp}_{adapt.}$ of the Adaptive Secrecy proof, where we show how consistent shares can be computed after generating t shares randomly. The (s_i) are the same in both distributions, given the $t+1$ points. In addition, all $\kappa_i, \{a_j^{(i)}, b_j^{(i)}\}$ are uniformly random. Therefore, the distributions are statistically indistinguishable.

D Supplementary Analysis

D.1 Time Limit Analysis of $\Pi_{\mathtt{P2P\text{-}Send}}(A, B, \mathtt{rid})$ in the Initial Attempt.

In the termination phase, the time limit $7\Delta + 2\mathtt{Time}_{\mathtt{bc}}$ is calculated as follows. When the network is synchronous and A is honest, after an honest party enters the $\mathtt{rid}$-th round,

- A will enter the $\mathtt{rid}$-th round within Δ time;
- A will wait for 2Δ time in the initial attempt;
- A will wait for $\mathtt{Time}_{\mathtt{bc}} + 4\Delta$ time in the resend attempt;
- A needs to broadcast the final message, costing at most $\mathtt{Time}_{\mathtt{bc}}$ time.

D.2 Time Limit Analysis of $\Pi_{\mathtt{P2P\text{-}Send}}(A, B, \mathtt{rid})$ in the Advanced Attempt.

In the termination phase, the time limit $3\Delta + \mathtt{Time}_{\mathtt{bc}} + n(4\Delta + \mathtt{Time}_{\mathtt{bc}})$ is calculated as follows. When the network is synchronous and A is honest, after an honest party enters the $\mathtt{rid}$-th round,

- A will enter the $\mathtt{rid}$-th round within Δ time;
- A will wait for 2Δ time in the initial attempt;
- A will wait for $\mathtt{Time}_{\mathtt{bc}} + 4\Delta$ time in each resend attempt, and the number of resend attempts is bounded by n in each round;
- A needs to broadcast the final message, costing at most $\mathtt{Time}_{\mathtt{bc}}$ time.

E Cost Analysis

E.1 Costs Analysis of $\Pi_{\mathtt{Channel}}$

In the following, we analyze the communication complexity of $\Pi_{\mathtt{Channel}}$. Let R be the number of rounds, $C_{\mathtt{p2p}}$ be the number of bits communicated via P2P channels among all parties, and $C_{\mathtt{bc}}$ be the number of bits broadcast via the broadcast channel. We assume that for every two parties (P_i, P_j) and in each round,

the number of bits sent from P_i to P_j is bounded by $C_{\mathtt{p2p}}(i,j)/n$, where $C_{\mathtt{p2p}}(i,j)$ is the number of bits communicated via the P2P channel from P_i to P_j. Note that this property can be achieved by separating the round that incurs a large amount of P2P communication into multiple rounds.

We will analyze all our three instantiations of $\mathcal{F}_{\mathsf{Channel}}$: the basic version described in Section 2.2, the advanced version described in Section 5.2, and the computationally secure version sketched in Section 5.2. In each instantiation, we will consider two cases: (1) the optimistic case where we assume the network is synchronous and all parties follow the protocol, and (2) the worst case where the network is asynchronous and corrupted parties may arbitrarily deviate from the protocol.

Communication Complexity of Our Basic Realization. In each round, all parties invoke $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$ for all parties $A, B \in \mathcal{P}$ and $\Pi_{\texttt{BC-Send}}(A, \texttt{rid})$ for all parties $A \in \mathcal{P}$. For the optimistic case,

1. Each call of $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$ requires B to send A the message m, costing $|m|$ bits of P2P communication. Then A broadcasts the confirmation $(\texttt{Confirm}, A, B, \texttt{rid})$, which is of $O(\log n + \log R)$ bits of broadcasting. Finally, all parties invoke $\Pi_{\texttt{BA}}$ where the input size is one bit per party.
2. Each call of $\Pi_{\texttt{BC-Send}}(A, \texttt{rid})$ requires A to broadcast message m, which is of $|m|$ bits. Then, all parties invoke $\Pi_{\texttt{BA}}$ where the input size is one bit per party.

Thus in total, all parties need to communicate $C_{\mathtt{p2p}}$ bits, broadcast $O(n^2 \cdot R)$ times and $C_{\mathtt{bc}} + O(n^2 \cdot R \cdot (\log n + \log R))$ bits, and invoke $\Pi_{\texttt{BA}}$ $O(n^2 \cdot R)$ times.

For the worst case,

1. Each call of $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$ requires B to send A the message m in the initial attempt, costing $|m|$ bits of P2P communication.
   Then A broadcasts the resend request $(\texttt{Resend}, A, B, \texttt{rid})$, which is of $O(\log n + \log R)$ bits of broadcasting. Next, B needs to send the message share to each intermediate with his signature, and each intermediate passes the message share to A. Note that for signatures generated by B, it is sufficient to restrict the verifiers to the intermediate party and A. However, for signatures generated by the intermediate party, we still need all parties to be able to verify them. In summary, this costs in total $O(n \cdot (|m| + \log n + \log R))$ bits plus $O(n)$ signatures of message size $O(n \cdot (|m| + \log n + \log R))$ bits for 3 verifiers and $O(n)$ signatures of message size $O(n(\log n + \log R))$ bits for n verifiers.
   After that, A may broadcast $(\texttt{Confirm}, A, B, \texttt{rid})$, $(\texttt{Eliminate}, A)$, or $(\texttt{Eliminate}, B, \texttt{Proof})$, which costs at most $O(\log n + \log R)$ bits plus $O(n)$ signatures of message size $O(n(\log n + \log R))$ bits for broadcasting. Finally, all parties invoke $\Pi_{\texttt{BA}}$ where the input size is one bit per party.
2. Each call of $\Pi_{\texttt{BC-Send}}(A, \texttt{rid})$ requires A to broadcast message m, which is of $|m|$ bits. Then, all parties invoke $\Pi_{\texttt{BA}}$ where the input size is one bit per party.

Thus in total, all parties need to sign $O(n^3 \cdot R)$ messages of message size $O(n \cdot C_{\mathtt{p2p}} + n^3 \cdot R \cdot (\log n + \log R))$ bits for 3 verifiers and $O(n^3 \cdot R)$ messages of message size $O(n^3 \cdot R \cdot (\log n + \log R))$ bits for n verifiers, communicate $O(n \cdot C_{\mathtt{p2p}} + n^3 \cdot R \cdot (\log n + \log R))$ bits plus $O(n^3 \cdot R)$ signatures on messages of total size $O(n \cdot C_{\mathtt{p2p}} + n^3 \cdot R \cdot (\log n + \log R))$ bits for 3 verifiers and $O(n^3 \cdot R)$ signatures on messages of total size $O(n^3 \cdot R \cdot (\log n + \log R))$ bits for n verifiers, broadcast $O(n^2 \cdot R)$ times and $C_{\mathtt{bc}} + O(n^2 \cdot R \cdot (\log n + \log R))$ bits plus $O(n^3 \cdot R)$ signatures on messages of total size $O(n^3 \cdot R \cdot (\log n + \log R))$ bits for n verifiers, and invoke $\Pi_{\texttt{BA}}$ $O(n^2 \cdot R)$ times.

Communication Complexity of Our Advanced Realization. The optimistic case remains the same as that in the basic version. Thus in total, all parties need to communicate $C_{\mathtt{p2p}}$ bits, broadcast $O(n^2 \cdot R)$ times and $C_{\mathtt{bc}} + O(n^2 \cdot R \cdot (\log n + \log R))$ bits, and invoke $\Pi_{\texttt{BA}}$ $O(n^2 \cdot R)$ times.

For the worst case, $\Pi_{\texttt{BC-Send}}(A, \texttt{rid})$ remains unchanged. As for $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$, we list the difference below.

1. The resend request becomes $(\texttt{Resend}, A, B, \texttt{ReCount}, \texttt{rid}, S)$, which is of size $O(n + \log R)$. During the resend, the cost becomes $O(t_a \cdot (|m| + \log n + \log R))$ bits plus $O(t_a)$ signatures of message size $O(t_a \cdot (|m| + \log n + \log R))$ for 3 verifiers and $O(t_a)$ signatures of message size $O(t_a \cdot (\log n + \log R))$ for n verifiers.

2. There are at most n failure resend attempts between every pair of parties A, B. Each failure attempt requires A to broadcast $O(n+\log R)$ bits, and all parties to communicate $O(t_a \cdot (|m|+\log n+\log R))$ bits plus $O(t_a)$ signatures of message size $O(t_a \cdot (|m| + \log n + \log R))$ for 3 verifiers and $O(t_a)$ signatures of message size $O(t_a \cdot (\log n + \log R))$ for n verifiers. The total cost (among all pairs of parties) is bounded by communicating $O(t_a \cdot C_{\mathtt{p2p}} + t_a \cdot n^3 \cdot (\log n + \log R))$ bits plus $O(t_a \cdot n^3)$ signatures on messages of total size $O(t_a \cdot C_{\mathtt{p2p}} + t_a \cdot n^3 \cdot (\log n + \log R))$ for 3 verifiers and $O(t_a \cdot n^3)$ signatures on messages of total size $O(t_a \cdot n^3 \cdot (\log n + \log R))$ for n verifiers, and broadcasting n^3 times of $O(n^3 \cdot (n + \log R))$ bits in total.

Thus in total, all parties need to sign $O(t_a \cdot n^2 \cdot (n+R))$ messages of size $O(t_a \cdot C_{\mathtt{p2p}} + t_a \cdot n^2 \cdot (n+R) \cdot (\log n + \log R))$ bits for 3 verifiers and $O(t_a \cdot n^2 \cdot (n+R))$ messages of size $O(t_a \cdot n^2 \cdot (n+R) \cdot (\log n + \log R))$ bits for n verifiers, communicate $O(t_a \cdot C_{\mathtt{p2p}} + t_a \cdot n^2 \cdot (n+R) \cdot (\log n + \log R))$ bits plus $O(t_a \cdot n^2 \cdot (n+R))$ signatures on messages of total size $O(t_a \cdot C_{\mathtt{p2p}} + t_a \cdot n^2 \cdot (n+R) \cdot (\log n + \log R))$ bits for 3 verifiers and $O(t_a \cdot n^2 \cdot (n+R))$ signatures on messages of total size $O(t_a \cdot n^2 \cdot (n+R) \cdot (\log n + \log R))$ bits for n verifiers, broadcast $O(n^2 \cdot (n+R))$ times and $C_{\mathtt{bc}} + O(n^2 \cdot (n+R) \cdot (n + \log R))$ bits plus $O(n^3 \cdot R)$ signatures on messages of total size $O(n^3 \cdot R \cdot (\log n + \log R))$ bits for n verifiers, and invoke $\Pi_{\mathtt{BA}}$ $O(n^2 \cdot R)$ times.

Communication Complexity of Our Computational Realization. Compared with the advanced variant, all parties will invoke $\Pi_{\mathtt{P2P\text{-}Send}}$ to exchange their symmetric keys in the beginning. Then the resend request size becomes $O(\log n + \log R)$ (since S only contains a single party), and the overhead is reduced from t_a to 1. In the computational setting, we do not specify the number of verifiers for each signature.

Thus, in the optimistic case, all parties need to communicate $C_{\mathtt{p2p}} + O(n^2 \cdot \kappa)$ bits, broadcast $O(n^2 \cdot R)$ times and $C_{\mathtt{bc}} + O(n^2 \cdot R \cdot (\log n + \log R))$ bits, and invoke $\Pi_{\mathtt{BA}}$ $O(n^2 \cdot R)$ times. In the worst case, all parties need to sign $O(n^2 \cdot (n+R))$ messages of size $O(C_{\mathtt{p2p}} + n^2 \cdot (n+R) \cdot (\log n + \log R) + n^3 \cdot \kappa)$ bits, communicate $O(C_{\mathtt{p2p}} + n^2 \cdot (n+R) \cdot (\log n + \log R) + n^3 \cdot \kappa)$ bits plus $O(n^2 \cdot (n+R))$ signatures on messages of total size $O(C_{\mathtt{p2p}} + n^2 \cdot (n+R) \cdot (\log n + \log R) + n^3 \cdot \kappa)$ bits, broadcast $O(n^2 \cdot (n+R))$ times and $C_{\mathtt{bc}} + O(n^2 \cdot (n+R) \cdot (\log n + \log R))$ bits plus $O(n^3 \cdot R)$ signatures on messages of total size $O(n^3 \cdot R \cdot (\log n + \log R))$ bits, and invoke $\Pi_{\mathtt{BA}}$ $O(n^2 \cdot R)$ times.

Summary. We summarize the final cost with both information-theoretic instantiations and computational instantiations of the signature schemes.

Variants	Setup Size	Settings	Communication Complexity
Basic Version	$O(nC_{\mathtt{p2p}} + n^2 C_{\mathtt{bc}} + n^6 R\kappa)$	Optimistic Case	$O(C_{\mathtt{p2p}} + n^4\kappa \cdot C_{\mathtt{bc}} + n^6 R\kappa \cdot \log(nR))$
		Worst Case	$O(nC_{\mathtt{p2p}} + n^4\kappa \cdot C_{\mathtt{bc}} + n^8\kappa^2 \cdot R)$
Advanced Version	$O(t_a C_{\mathtt{p2p}} + n^2 C_{\mathtt{bc}} + n^6(n+R)\kappa)$	Optimistic Case	$O(C_{\mathtt{p2p}} + n^4\kappa \cdot C_{\mathtt{bc}} + n^6(n+R)\kappa \cdot \log(nR))$
		Worst Case	$O(t_a C_{\mathtt{p2p}} + n^4\kappa \cdot C_{\mathtt{bc}} + n^8\kappa^2 \cdot (n+R))$
Computational Version	$O(n^2 \cdot \kappa)$	Optimistic Case	$O(C_{\mathtt{p2p}} + n^4\kappa \cdot C_{\mathtt{bc}} + n^6(n+R)\kappa \cdot \log(nR))$
		Worst Case	$O(C_{\mathtt{p2p}} + n^4\kappa \cdot C_{\mathtt{bc}} + n^7\kappa^2 \cdot (n+R))$

Fig. 1. Cost of $\Pi_{\mathtt{Channel}}$ with Different Instantiations

E.2 Cost Analysis of Our Compiler with Different Instantiations

Recall that our compiler requires the following primitives:

- A (t_s, n)-robust secret sharing scheme with adaptive secrecy, online reconstruction, and efficient resampling;
- A synchronous MPC $\Pi_{\mathtt{Input}}$ with GOD that securely computes $\mathcal{F}_{\mathsf{Input}}$ against t_s corrupted parties with adaptive security assuming a broadcast channel;
- A synchronous MPC with GOD $\Pi_{\mathtt{Comp}}$ that securely computes $\mathcal{F}_{\mathsf{Comp}}$ against t_s corrupted parties with adaptive security assuming a broadcast channel;
- An asynchronous MPC $\Pi_{\mathtt{AMPC\text{-}GOD}}$ with GOD that securely computes $\mathcal{F}_{\mathsf{AMPC}}$ among $t_s + 2t_a + 1$ parties against t_a corrupted parties such that when running in the synchronous network, all honest parties' inputs will be considered.

For an n-ary function f with function size $|C|$, input size C_I, and output size C_O (Here C_O is the summation of the output size of all parties). The cost of our compiler consists of the following parts.

- Invocation of $\mathcal{F}_{\mathsf{Channel}}$: All parties need to invoke $\mathcal{F}_{\mathsf{Channel}}$ for $R = nR_I + R_C$ rounds. The total P2P message size $C_{\mathtt{p2p}}$ is bounded by the summation of the P2P message size of each $\Pi_{\mathtt{Input}}(D)$ and $\Pi_{\mathtt{Comp}}$. The total broadcast size $C_{\mathtt{bc}}$ is bounded by the summation of the broadcast message size of each $\Pi_{\mathtt{Input}}(D)$ and $\Pi_{\mathtt{Comp}}$.
- Reconstruction phase: The communication is bounded by C_O times of the sharing size of a robust secret sharing.
- Fallback Procedure: The communication is bounded by that of $\Pi_{\mathtt{AMPC\text{-}GOD}}$ plus $nC_O\kappa$ bits.

Instantiations of Different Primitives. For the robust secret sharing scheme, we use the scheme introduced in Section 4.4, which has sharing size $O(n^2\kappa)$ bits.

For synchronous MPCs with GOD and adaptive security, we use the MPC protocol from [BSFO12, GSZ20], which requires $O(n|C|\kappa + n^6\kappa)$ communication, $O(n^3\kappa)$ broadcast size, and $O(D + n^2)$ rounds where $|C|$ is the function size and D is the function depth. Then $\mathcal{F}_{\mathsf{Input}}(D)$ has function size $n^2C_{I,D}$ and function depth $O(n)$, where $C_{I,D}$ is the input size of D. We obtain $\Pi_{\mathtt{Input}}$ with communication complexity $O(n^3C_{I,D}\kappa + n^6\kappa)$, broadcast size $O(n^3\kappa)$, and round complexity $O(n^2)$. $\mathcal{F}_{\mathsf{Comp}}$ has function size $|C| + n^2C_I + n^2C_O$ and function depth $O(D + n)$, where $|C|$ is the function size of f, D is the function depth of f, $C_I = \sum_D C_{I,D}$ is the input size, and C_O is the function size. We obtain $\Pi_{\mathtt{Comp}}$ with communication complexity $O(n|C|\kappa + n^3C_I\kappa + n^3C_O\kappa + n^6\kappa)$, broadcast size $O(n^3\kappa)$, and round complexity $O(D + n^2)$.

Finally, for $\Pi_{\mathtt{AMPC\text{-}GOD}}$, we use the result from [GLZS24, JLS24] which achieves communication complexity $O(n|C|\kappa + n^2D\kappa + n^{14}\kappa^2)$. Recall that our compiler requires $\Pi_{\mathtt{AMPC\text{-}GOD}}$ to satisfy that when running in the synchronous setting, all honest parties' inputs will be used in the computation. We briefly discuss how this can be achieved from the MPC protocol in [GLZS24, JLS24].

We first show that for any asynchronous protocol Π which satisfies that with overwhelming probability, the communication is bounded by Com and all honest parties terminate, when running Π in the synchronous network, there is a time bound T such that with overwhelming probability, all honest parties terminate Π within time bound T. To see this, note that the asynchronous protocol Π is driven by message. Consider an execution of Π in the synchronous network, if there exists a time T' such that

1. Not all honest parties terminate,
2. All honest parties' messages have been delivered,
3. Honest parties who do not terminate are waiting for messages,

then corrupted parties can choose to not send any message and all honest parties will never terminate. Since the synchronous network can be seen as a special case of the asynchronous network where the adversary just schedules the delivery of messages following the synchronous network, if with non-negligible probability, such T' exists, it implies that with non-negligible probability, Π does not terminate, which contradicts with the assumption. Thus, with overwhelming probability, at each time T', at least one of the following three conditions happen:

1. All honest parties terminate;

2. Some honest party's message has not been delivered;
3. Some honest party is computing the messages that should be sent to other parties.

Let T_r be the maximum computation time of an honest party. Consider the total delivered communication between honest parties at time T'. If not all honest parties terminate at time T', then after time $T_r + \Delta$, the total communication between honest parties will be increased by at least 1 bit. This is because for the second case, the honest party's message will be delivered within $T' + \Delta$, and for the third case, the honest party will send his message within $T' + T_r$ and the message will be delivered within $T' + T_r + \Delta$. Since the total communication is bounded by Com, the protocol Π will terminate within time $(T_r + \Delta) \cdot \mathsf{Com}$ with overwhelming probability.

Come back to the asynchronous MPC protocol in [GLZS24, JLS24]. Their construction idea is to let all parties compute a degree-t Shamir sharing of every wire in the computation. In their input phase, each party uses an ACSS (asynchronous complete secret sharing) protocol to share his input using degree-t Shamir sharings. Then all parties run an ACS (agreement on a common subset) protocol to agree on a set Core of at least $n - t$ parties that successfully share their inputs. Then the protocol will compute the function output using inputs from parties in Core. Thus, to achieve our goal, it is sufficient to let Core contain all honest parties. The change is as follows. In the beginning, all parties first run the following modified input phase.

1. Each party uses an ACSS protocol with guarantee of termination to share his input.
2. Let T be the time bound for the ACSS protocol when running in the synchronous network. Each party moves on to the next step when either (1) he has terminated the ACSS protocol for all parties, or (2) he has waited for time T.
3. All parties run the ACS protocol that are instantiated by n instances of BA [BKR94]:
   (a) Each party P_i sets his input of the j-th BA to be 1 if he has terminated the ACSS led by P_j.
   (b) If P_i has received 1 as output for at least $n - t$ instances of BA, then for each BA where P_i has not decided his input, he sets his input to be 0.

   All parties take the set of all parties whose corresponding BA outputs 1.

Note that when the above process runs in the synchronous network, the ACSS protocol led by each honest party P_i will terminate within time T with overwhelming probability, and thus P_i's input will be taken into consideration. When instantiating the ACSS protocol from [JLS24], the input phase requires $O(n^2 C_I \kappa + n^{14}\kappa^2)$ communication.

F Security Proofs

F.1 Proof of Lemma 1

Let $\mathtt{Time}_{\mathsf{bc}}$ be the running time of $\Pi_{\mathtt{RBC}}$ when the network is synchronous and the dealer is honest, and $\mathtt{Time}_{\mathsf{ba}}$ be the running time of $\Pi_{\mathtt{BA}}$ when the network is synchronous and the starting time of all honest parties differs by Δ at most. We set $\tau_{\mathsf{synch}} = 3\Delta + n(4\Delta + \mathtt{Time}_{\mathsf{bc}}) + \mathtt{Time}_{\mathsf{ba}}$. We start with the construction of the simulator $\mathcal{S}$. Let $\mathcal{E}$ be the environment and $\mathcal{A}$ be the real world adversary. $\mathcal{S}$ receives from $\mathcal{E}$ the party set $\mathcal{P} = \{P_1, \ldots, P_n\}$, the set of corrupted parties $\mathcal{C}orr$, corruption threshold t_s, t_a, the delivery time-bound Δ for a message sent in the synchronous network, the network configuration $\mathsf{mode} \in \{\mathsf{synch}, \mathsf{asynch}\}$, and a set $\mathcal{R}$ of valid round ids.

$\mathcal{S}$ runs $\mathcal{A}$ and honestly passes messages between $\mathcal{E}$ and $\mathcal{A}$. $\mathcal{S}$ also invokes $\mathcal{F}_{\mathsf{Channel}}$ with the inputs required by $\mathcal{F}_{\mathsf{Channel}}$. In the following $\mathcal{S}$ will simulate the behaviors of honest parties.

In the initialization phase, $\mathcal{S}$ honestly emulates $\mathcal{F}_{\mathsf{Setup}}$ to generate $(\mathsf{sk}^{(i)}, \{\mathsf{vk}_j^{(i)}\}_{j=1}^n)$ for each party $P_i \in \mathcal{P}$ and distribute the keys to corrupted parties. $\mathcal{S}$ sets $R = |\mathcal{R}|$.

In the online phase, whenever an honest party P_i receives the request $(\mathtt{rid}, m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ from $\mathcal{E}$, $\mathcal{S}$ receives $(P_i, \mathtt{rid}, m_0^{(i)}, \{m_j^{(i)}\}_{j \in \mathcal{C}orr})$ from $\mathcal{F}_{\mathsf{Channel}}$. If $\mathsf{mode} = \mathsf{synch}$, $\mathcal{S}$ also receives the receiving time τ. $\mathcal{S}$ follows $\mathcal{F}_{\mathsf{Channel}}$ to update $\mathtt{wID}, \mathcal{U}$ and records $\tau_{\mathtt{rid}}$.

Then $\mathcal{S}$ invokes $\Pi_{\mathtt{P2P\text{-}Send}}(A, B, \mathtt{rid})$ for every $A, B \in \mathcal{P}$ and invokes $\Pi_{\mathtt{BC\text{-}Send}}(A, \mathtt{rid})$ for every $A \in \mathcal{P}$ on behalf of P_i. For $\Pi_{\mathtt{P2P\text{-}Send}}(A, B, \mathtt{rid})$ and $\Pi_{\mathtt{BC\text{-}Send}}(A, \mathtt{rid})$, $\mathcal{S}$ honestly follows the protocol to simulate the behaviors of P_i with the following change: When $B = P_i$ and $A = P_j$ is an honest party,

– If $\mathcal{F}_{\mathsf{Channel}}$ has given up the security, $\mathcal{S}$ learns $m_j^{(i)}$ from $\mathcal{F}_{\mathsf{Channel}}$ and uses it in $\Pi_{\texttt{P2P-Send}}(P_j, P_i, \texttt{rid})$;
– Otherwise, $\mathcal{S}$ uses an all-0 string as $m_j^{(i)}$ in $\Pi_{\texttt{P2P-Send}}(P_j, P_i, \texttt{rid})$.

During the simulation, whenever $\mathcal{S}$ receives a message signed by an honest party from corrupted parties, $\mathcal{S}$ checks whether this signature is generated by $\mathcal{S}$. If not, $\mathcal{S}$ aborts and the output is replaced by $\perp$.

For each $\texttt{rid}$, upon the first honest party terminating $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$ for all $A, B \in \mathcal{P}$ and $\Pi_{\texttt{BC-Send}}(A, \texttt{rid})$ for all $A \in \mathcal{P}$, $\mathcal{S}$ follows the protocol to decide W. Then for each corrupted party $P_i \notin W$, $\mathcal{S}$ learns $(m_0^{(i)}, \{m_j^{(i)}\}_{j \notin W \cup \mathcal{C}orr})$. $\mathcal{S}$ sets $m_j^{(i)} = \mathbf{0}$ for all $P_j \in W \cup \mathcal{C}orr$. Then $\mathcal{S}$ sends $(\texttt{rid}, m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ to $\mathcal{F}_{\mathsf{Channel}}$ on behalf of P_i. Next $\mathcal{S}$ sends $(\mathsf{Proceed}, \texttt{rid}, W)$ to $\mathcal{F}_{\mathsf{Channel}}$. Finally, upon each honest party P_j terminating $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$ for all $A, B \in \mathcal{P}$ and $\Pi_{\texttt{BC-Send}}(A, \texttt{rid})$ for all $A \in \mathcal{P}$, $\mathcal{S}$ lets $\mathcal{F}_{\mathsf{Channel}}$ deliver the output of P_j for the $\texttt{rid}$-th round to P_j.

If $\mathsf{mode} = \mathsf{synch}$ and $\mathcal{F}_{\mathsf{Channel}}$ gives up the security, upon each honest party P_j terminating $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$ for all $A, B \in \mathcal{P}$ and $\Pi_{\texttt{BC-Send}}(A, \texttt{rid})$ for all $A \in \mathcal{P}$, $\mathcal{S}$ learns $\{m_j^{(i)}\}_{j \notin W \cup \mathcal{C}orr}$ from $\mathcal{F}_{\mathsf{Channel}}$. If $P_j \in W$, $\mathcal{S}$ delivers $(\texttt{rid}, W, \varnothing)$ to P_j on behalf of $\mathcal{F}_{\mathsf{Channel}}$. If $P_j \notin W$, $\mathcal{S}$ delivers $(\texttt{rid}, W, \{(i, m_0^{(i)}, m_j^{(i)})\}_{i \notin W})$ on behalf of $\mathcal{F}_{\mathsf{Channel}}$.

Finally, $\mathcal{S}$ outputs the view of $\mathcal{A}$.

Now we show that any computationally bounded/unbounded (depending on the signature scheme used in the construction) environment $\mathcal{E}$ cannot distinguish the real world and the ideal world.

$\mathbf{Hyb}_0$: In the initial hybrid, we consider the real world execution and define the output to be the output of $\mathcal{E}$.

$\mathbf{Hyb}_1$: In this hybrid, whenever $\mathcal{S}$ receives a message signed by an honest party from corrupted parties, $\mathcal{S}$ checks whether this signature is generated by $\mathcal{S}$. If not, $\mathcal{S}$ aborts and the output is replaced by $\perp$. By the security of the signature scheme, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ are statistically close when an information-theoretic signature scheme is used, and computationally close when a computationally secure signature scheme is used.

$\mathbf{Hyb}_2$: In this hybrid, for each $\texttt{rid}$, upon the first honest party terminating $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$ for all $A, B \in \mathcal{P}$ and $\Pi_{\texttt{BC-Send}}(A, \texttt{rid})$ for all $A \in \mathcal{P}$, $\mathcal{S}$ follows the protocol to decide W. Then for each corrupted party $P_i \notin W$, $\mathcal{S}$ learns $(m_0^{(i)}, \{m_j^{(i)}\}_{j \notin W \cup \mathcal{C}orr})$. $\mathcal{S}$ sets $m_j^{(i)} = \mathbf{0}$ for all $P_j \in W \cup \mathcal{C}orr$. Note that this does not change the output distribution.

We show that for each honest party P_j, after P_j terminates $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$ for all $A, B \in \mathcal{P}$ and $\Pi_{\texttt{BC-Send}}(A, \texttt{rid})$ for all $A \in \mathcal{P}$,

– If $P_j \in W$, then P_j would output $(\texttt{rid}, W, \varnothing)$.
– If $P_j \notin W$, then P_j would output $(\texttt{rid}, W, \{(i, m_0^{(i)}, m_j^{(i)})\}_{i \notin W})$.

By the security of $\Pi_{\texttt{BA}}$ and $\Pi_{\texttt{RBC}}$, all honest parties would obtain the same output for each call of $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$ and $\Pi_{\texttt{BC-Send}}(A, \texttt{rid})$. Thus all honest parties would obtain the same set W and the same broadcast messages. As for P2P messages, when $\mathcal{S}$ does not abort, it is guaranteed in $\mathbf{Hyb}_1$ that an honest P_j would obtain the correct $m_j^{(i)}$ from P_i.

$\mathbf{Hyb}_3$: In this hybrid, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Channel}}$ honestly. For each honest P_i, upon receiving $(\texttt{rid}, m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ from $\mathcal{E}$, $\mathcal{S}$ sends it to $\mathcal{F}_{\mathsf{Channel}}$. For each corrupted P_i, after $\mathcal{S}$ determined $(\texttt{rid}, m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ as described in $\mathbf{Hyb}_2$, $\mathcal{S}$ sends it to $\mathcal{F}_{\mathsf{Channel}}$. After $\mathcal{S}$ decided W as described in $\mathbf{Hyb}_2$, $\mathcal{S}$ sends $(\mathsf{Proceed}, \texttt{rid}, W)$ to $\mathcal{F}_{\mathsf{Channel}}$.

Note that we do not use the output from $\mathcal{F}_{\mathsf{Channel}}$. This does not change the output distribution.

$\mathbf{Hyb}_4$: In this hybrid, for each pair of honest parties P_i, P_j,

– If $\mathcal{F}_{\mathsf{Channel}}$ has given up the security, $\mathcal{S}$ uses the actual $m_j^{(i)}$ in $\Pi_{\texttt{P2P-Send}}(P_j, P_i, \texttt{rid})$;
– Otherwise, $\mathcal{S}$ uses an all-0 string as $m_j^{(i)}$ in $\Pi_{\texttt{P2P-Send}}(P_j, P_i, \texttt{rid})$, but sets the output of P_j to be the actual $m_j^{(i)}$ if the final output of $\Pi_{\texttt{P2P-Send}}(P_i, P_j, \texttt{rid})$ is $(\texttt{Confirm}, P_j, P_i, \texttt{rid})$.

We argue that $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ are identically distributed. First note that if $\mathcal{F}_{\mathsf{Channel}}$ gives up the security, the execution remains the same as that in $\mathbf{Hyb}_3$. When $\mathcal{F}_{\mathsf{Channel}}$ does not give up the security, there are two cases.

If $\mathsf{mode} = \mathsf{synch}$, it implies that the time difference between P_i and P_j entering the $\mathtt{rid}$-th round is bounded by Δ. In this case, P_j would receive $m_j^{(i)}$ from P_i in the initial attempt via the point-to-point communication channel. By the security of the communication channel, $\mathcal{A}$ learns no information about the message sent between a pair of honest parties.

If $\mathsf{mode} = \mathsf{asynch}$, then $|\mathcal{C}orr| \leq t_a$. Corrupted parties only learn at most t_a shares of $[m_j^{(i)}]_{t_a}$. By the security of Shamir secret sharings, these t_a shares are independent of $m_j^{(i)}$. Furthermore, since P_j will only accept the correct shares from P_i due to the signatures, when the final output of $\Pi_{\mathtt{P2P\text{-}Send}}(P_i, P_j, \mathtt{rid})$ is $(\mathtt{Confirm}, P_j, P_i, \mathtt{rid})$, P_j would output $m_j^{(i)}$ in $\mathbf{Hyb}_3$.

Thus, in either case, $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ are identically distributed.

Now we argue that when $\mathsf{mode} = \mathsf{synch}$, after $\mathcal{S}$ has decided W for $\mathtt{rid}$ at time τ and sent $(\mathsf{Proceed}, \mathtt{rid}, W)$ to $\mathcal{F}_{\mathsf{Channel}}$, if $\mathcal{F}_{\mathsf{Channel}}$ has not given up the security, then (1) $\mathtt{wID} = \mathtt{rid}$, (2) $\tau \leq \tau_{\mathtt{rid}} + \tau_{\mathsf{synch}}$, (3) $W \subset \mathcal{C}orr$, and (4) for each $P_i \notin W$, $\mathcal{S}$ has obtained $(\mathtt{rid}, m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$. Furthermore, all honest parties would terminate $\Pi_{\mathtt{P2P\text{-}Send}}(A, B, \mathtt{rid})$ for all $A, B \in \mathcal{P}$ and $\Pi_{\mathtt{BC\text{-}Send}}(A, \mathtt{rid})$ for all $A \in \mathcal{P}$ before $\tau + \Delta$.

First, note that at least one honest party has terminated $\Pi_{\mathtt{P2P\text{-}Send}}(A, B, \mathtt{rid})$ for all $A, B \in \mathcal{P}$ and $\Pi_{\mathtt{BC\text{-}Send}}(A, \mathtt{rid})$ for all $A \in \mathcal{P}$ at time τ, which means that (1) all honest parties would terminate $\Pi_{\mathtt{P2P\text{-}Send}}$ and $\Pi_{\mathtt{BC\text{-}Send}}$ within $\tau + \Delta$, and (2) $\mathcal{S}$ has provided the request from this party to $\mathcal{F}_{\mathsf{Channel}}$. By construction, $\mathcal{F}_{\mathsf{Channel}}$ has set $\tau_{\mathtt{rid}}$ and marked $\mathtt{wID} = \mathtt{rid}$. Also by construction, $\mathcal{F}_{\mathsf{Channel}}$ will only reset $\mathtt{rid}$ when either receiving $(\mathsf{Proceed}, W, \mathtt{rid})$ or at time $\tau_{\mathtt{rid}} + \tau_{\mathsf{synch}}$. We consider two cases.

Case 1: $\tau \leq \tau_{\mathtt{rid}} + \Delta$. In this case, since $\mathcal{F}_{\mathsf{Channel}}$ has not given up the security, (1) and (2) hold. We also have that $\mathcal{S}$ has received the request for each honest party P_i from $\mathcal{E}$ and have provided them to $\mathcal{F}_{\mathsf{Channel}}$. This is because an honest party terminating $\Pi_{\mathtt{BC\text{-}Send}}(P_i, \mathtt{rid})$ at time $\tau \leq \tau_{\mathtt{rid}} + \Delta$ implies that this honest party has received the broadcast message from P_i (or otherwise, he would wait for at least $\Delta + \mathtt{Time}_{\mathsf{bc}}$ time before terminating $\Pi_{\mathtt{BC\text{-}Send}}(P_i, \mathtt{rid})$), which further implies that P_i has received the request from $\mathcal{E}$ before $\tau_{\mathtt{rid}} + \Delta$.

Note that the time difference among honest parties entering the $\mathtt{rid}$-th round is bounded by Δ. From our analysis about $\Pi_{\mathtt{BC\text{-}Send}}$ and $\Pi_{\mathtt{P2P\text{-}Send}}$, an honest party would not be identified. Thus, $W \subset \mathcal{C}orr$. Since $\mathcal{S}$ has provided the request of each corrupted party $P_i \notin W$ to $\mathcal{F}_{\mathsf{Channel}}$ as well, (3) and (4) hold.

Case 2: $\tau > \tau_{\mathtt{rid}} + \Delta$. In this case, since $\mathcal{F}_{\mathsf{Channel}}$ has not given up the security, it means that $\mathcal{S}$ has provided the requests of all honest parties to $\mathcal{F}_{\mathsf{Channel}}$ within time $\tau_{\mathtt{rid}} + \Delta$. This implies that the time difference among honest parties entering the $\mathtt{rid}$-th round is bounded by Δ. Thus, from our analysis about $\Pi_{\mathtt{BC\text{-}Send}}$ and $\Pi_{\mathtt{P2P\text{-}Send}}$, an honest party would not be identified, implying that $W \subset \mathcal{C}orr$. Since $\mathcal{S}$ has provided the request of each corrupted party $P_i \notin W$ to $\mathcal{F}_{\mathsf{Channel}}$ as well, (3) and (4) hold.

As for (1) and (2), it is sufficient to show that $\tau \leq \tau_{\mathtt{rid}} + \tau_{\mathsf{synch}}$. Again since the time difference among honest parties entering the $\mathtt{rid}$-th round is bounded by Δ, from our analysis about $\Pi_{\mathtt{BC\text{-}Send}}$ and $\Pi_{\mathtt{P2P\text{-}Send}}$, an honest party would terminate $\Pi_{\mathtt{BC\text{-}Send}}$ and $\Pi_{\mathtt{P2P\text{-}Send}}$ before $\tau_{\mathtt{rid}} + \tau_{\mathsf{synch}}$.

$\mathbf{Hyb}_5$: In this hybrid, we consider the ideal world scenario. Essentially, we let honest parties take the outputs received from $\mathcal{F}_{\mathsf{Channel}}$. As we have shown in $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_4$, when $\mathcal{F}_{\mathsf{Channel}}$ does not give up the security, it would generate the same outputs as those of honest parties in $\mathbf{Hyb}_4$. When $\mathcal{F}_{\mathsf{Channel}}$ gives up the security, it remains the same as $\mathbf{Hyb}_4$ since the outputs of honest parties are determined by $\mathcal{S}$.

Thus, $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ are identically distributed.

F.2 Security Proof of Lemma 2

We set $\tau_0 = \tau_f$ to be the running time of $\Pi_{\mathtt{Compiler}}$ when the network is synchronous. We start with the construction of the simulator $\mathcal{S}$. Let $\mathcal{E}$ be the environment and $\mathcal{A}$ be the real-world adversary. $\mathcal{S}$ receives from $\mathcal{E}$ the party set $\mathcal{P} = \{P_1, \ldots, P_n\}$, the set of corrupted parties $\mathcal{C}orr$, and the network configuration $\mathsf{mode} \in \{\mathsf{synch}, \mathsf{asynch}\}$.

$\mathcal{S}$ runs $\mathcal{A}$ and honestly passes messages between $\mathcal{E}$ and $\mathcal{A}$. $\mathcal{S}$ also invokes $\mathcal{F}_{\mathsf{sfe}}$. In the following, $\mathcal{S}$ will simulate the behaviors of honest parties and $\mathcal{F}_{\mathsf{Channel}}$.

$\mathcal{S}$ honestly emulates $\mathcal{F}_{\mathsf{Channel}}$. Whenever $\mathcal{S}$ receives (Input, P_i) for an honest party P_i, $\mathcal{S}$ starts simulating the party P_i.

To simulate the input phase, let $\mathcal{A}_{\mathtt{input}}$ be the adversary for the underlying synchronous MPC protocol $\Pi_{\mathtt{Input}}$ who just follows the instructions received from the environment and passes the messages received from honest parties in each round. Let $\mathcal{S}_{\mathtt{input}}$ be the simulator for $\mathcal{A}_{\mathtt{input}}$. For every $D \in \{P_1, \ldots, P_n\} \backslash \mathcal{W}$, $\mathcal{S}$ acts as $\mathcal{E}_{\mathtt{input}}$ and $\mathcal{F}_{\mathsf{Input}}(D)$ and invokes $\mathcal{S}_{\mathtt{input}}$. $\mathcal{S}$ provides to D an all-0 string as D's input. For $\mathcal{F}_{\mathsf{Input}}(D)$,

1. $\mathcal{S}$ receives $\mathcal{C}orr'$ from $\mathcal{S}_{\mathtt{input}}$. If $D \in \mathcal{C}orr'$, $\mathcal{S}$ also receives the input s from $\mathcal{S}_{\mathtt{input}}$. Otherwise, $\mathcal{S}$ sets $s = 0$.
2. $\mathcal{S}$ generates a random robust secret sharing $[\![s]\!]$ and sends the shares of parties in $\mathcal{C}orr'$ to $\mathcal{S}_{\mathtt{input}}$.
3. Whenever receiving $(\mathtt{corrupt}, P_i)$ from $\mathcal{S}_{\mathtt{input}}$, if $P_i \notin \mathcal{C}orr'$ and $|\mathcal{C}orr'| < t_s$, $\mathcal{S}$ adds P_i in $\mathcal{C}orr'$ and sends P_i's share of $[\![s]\!]$ to $\mathcal{S}_{\mathtt{input}}$.

For $\mathcal{E}_{\mathtt{input}}$, $\mathcal{S}$ instructs $\mathcal{A}_{\mathtt{input}}$ to corrupt parties in $\mathcal{C}orr' = \mathcal{C}orr \cup \mathcal{W}$. Then from $\mathtt{rid} = 1$ to R_I,

1. $\mathcal{S}$ receives from $\mathcal{A}_{\mathtt{input}}$ P2P messages and broadcast messages that parties in $\mathcal{P} \backslash \mathcal{C}orr'$ send to parties in $\mathcal{C}orr'$.
2. In $\Pi_{\mathtt{Compiler}}$, for each honest party $P_i \notin \mathcal{W}$, after P_i enters the $\mathtt{rid}$-th round, $\mathcal{S}$ prepares $(m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ as follows: $\mathcal{S}$ sets $m_0^{(i)}$ as the broadcast message of P_i received from $\mathcal{A}_{\mathtt{input}}$. For each $P_j \in \mathcal{C}orr'$, $\mathcal{S}$ sets $m_j^{(i)}$ as the P2P message sent from P_i to P_j received from $\mathcal{A}_{\mathtt{input}}$. For each party $P_j \in \mathcal{P} \backslash \mathcal{C}orr'$, $\mathcal{S}$ sets $m_j^{(i)}$ to be an all-0 string. Then $\mathcal{S}$ sends $((\Pi_{\mathtt{Input}}, D, \mathtt{rid}), m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ to $\mathcal{F}_{\mathsf{Channel}}$ on behalf of P_i.
3. In $\Pi_{\mathtt{Compiler}}$, for each honest party $P_i \in \mathcal{W}$, after P_i enters the $\mathtt{rid}$-th round, $\mathcal{S}$ sets $(m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ as all-0 strings. Then $\mathcal{S}$ sends $((\Pi_{\mathtt{Input}}, D, \mathtt{rid}), m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ to $\mathcal{F}_{\mathsf{Channel}}$ on behalf of P_i.
4. In $\Pi_{\mathtt{Compiler}}$, after $\mathcal{F}_{\mathsf{Channel}}$ has delivered the $(\Pi_{\mathtt{Input}}, D, \mathtt{rid})$-th round message to an honest party, $\mathcal{S}$ learns from $\mathcal{F}_{\mathsf{Channel}}$ the set W' and updates $\mathcal{W} := \mathcal{W} \cup W'$.
   - If $D \in \mathcal{W}$ or $|\mathcal{W}| > t_s - t_a$, $\mathcal{S}$ instructs $\mathcal{A}_{\mathtt{input}}$ to set all messages sent from parties in $\mathcal{C}orr'$ to parties in $\mathcal{P} \backslash \mathcal{C}orr'$ to be all-0 strings in the following rounds of $\Pi_{\mathtt{Input}}$. Then $\mathcal{S}$ breaks the loop.
   - If $\mathsf{mode} = \mathsf{asynch}$, for each party P_i in $\mathcal{W}$ that is not corrupted by $\mathcal{A}_{\mathtt{input}}$ in $\Pi_{\mathtt{Input}}$, $\mathcal{S}$ instructs $\mathcal{A}_{\mathtt{input}}$ to corrupt P_i. Then $\mathcal{S}$ updates $\mathcal{C}orr' = \mathcal{C}orr \cup \mathcal{W}$.
5. In $\Pi_{\mathtt{Input}}$, for each party $P_i \in \mathcal{W}$, $\mathcal{S}$ instructs $\mathcal{A}_{\mathtt{input}}$ to set all P_i's messages sent/broadcast in the $\mathtt{rid}$-th round to be all-0 strings. For each party $P_i \in \mathcal{C}orr \backslash \mathcal{W}$, $\mathcal{S}$ instructs $\mathcal{A}_{\mathtt{input}}$ to set P_i's messages to be the same as those received by $\mathcal{F}_{\mathsf{Channel}}$.

If $\mathcal{S}$ finishes the loop, $\mathcal{S}$ adds D in $\mathcal{I}$ and moves on to the next dealer. If $\mathcal{S}$ breaks the loop due to $D \in \mathcal{W}$, $\mathcal{S}$ moves on to the next dealer. If $\mathcal{S}$ breaks the loop due to $|\mathcal{W}| > t_s - t_a$, $\mathcal{S}$ moves to the fallback procedure.

To simulate the computation phase, let $\mathcal{A}_{\mathtt{comp}}$ be the adversary for the underlying synchronous MPC protocol $\Pi_{\mathtt{Comp}}$ who just follows the instructions received from the environment and passes the messages received from honest parties in each round. Let $\mathcal{S}_{\mathtt{comp}}$ be the simulator for $\mathcal{A}_{\mathtt{comp}}$. $\mathcal{S}$ acts as $\mathcal{E}_{\mathtt{comp}}$ and $\mathcal{F}_{\mathsf{Comp}}$ and invokes $\mathcal{S}_{\mathtt{comp}}$. $\mathcal{S}$ provides to all parties the set $\mathcal{I}$ and all-0 strings as their shares of $\{[\![x_i]\!]\}_{P_i \in \mathcal{I}}$. For $\mathcal{F}_{\mathsf{Comp}}$,

1. $\mathcal{S}$ receives $\mathcal{C}orr'$ from $\mathcal{S}_{\mathtt{comp}}$.
2. $\mathcal{S}$ receives the set $\mathcal{I} \subset \mathcal{P}$. Recall that for each $P_i \in \mathcal{I}$, $\mathcal{S}$ has generated a random robust secret sharing $[\![x_i]\!]$ when simulating $\mathcal{F}_{\mathsf{Input}}$.
3. $\mathcal{S}$ sets $y = 0$ and generates n random robust secret sharings of y, denoted by $\{[\![y_i]\!]\}_{i=1}^n$. Then $\mathcal{S}$ sends the shares of $\{[\![y_i]\!]\}_{i=1}^n$ of parties in $\mathcal{C}orr'$ to $\mathcal{S}_{\mathtt{comp}}$.
4. Whenever receiving $(\mathtt{corrupt}, P_i)$ from $\mathcal{S}_{\mathtt{comp}}$, if $P_i \notin \mathcal{C}orr'$ and $|\mathcal{C}orr'| < t_s$, $\mathcal{S}$ adds P_i in $\mathcal{C}orr'$ and sends P_i's shares of $\{[\![x_i]\!]\}_{P_i \in \mathcal{I}} \cup \{[\![y_i]\!]\}_{i=1}^n$ to $\mathcal{S}_{\mathtt{comp}}$.

For $\mathcal{E}_{\mathtt{comp}}$, $\mathcal{S}$ instructs $\mathcal{A}_{\mathtt{comp}}$ to corrupt parties in $\mathcal{C}orr' := \mathcal{C}orr \cup \mathcal{W}$. Then from $\mathtt{rid} = 1$ to R_C,

1. $\mathcal{S}$ receives from $\mathcal{A}_{\mathtt{comp}}$ P2P messages and broadcast messages that parties in $\mathcal{P}\backslash\mathcal{C}orr'$ send to parties in $\mathcal{C}orr'$.
2. In $\Pi_{\mathtt{Compiler}}$, for each honest party $P_i \notin \mathcal{W}$, after P_i enters the $\mathtt{rid}$-th round, $\mathcal{S}$ prepares $(m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ as follows: $\mathcal{S}$ sets $m_0^{(i)}$ as the broadcast message of P_i received from $\mathcal{A}_{\mathtt{comp}}$. For each $P_j \in \mathcal{C}orr'$, $\mathcal{S}$ sets $m_j^{(i)}$ as the P2P message sent from P_i to P_j received from $\mathcal{A}_{\mathtt{comp}}$. For each party $P_j \in \mathcal{P}\backslash\mathcal{C}orr'$, $\mathcal{S}$ sets $m_j^{(i)}$ to be an all-0 string. Then $\mathcal{S}$ sends $((\Pi_{\mathtt{Comp}}, \mathtt{rid}), m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ to $\mathcal{F}_{\mathsf{Channel}}$ on behalf of P_i.
3. In $\Pi_{\mathtt{Compiler}}$, for each honest party $P_i \in \mathcal{W}$, after P_i enters the $\mathtt{rid}$-th round, $\mathcal{S}$ sets $(m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ as all-0 strings. Then $\mathcal{S}$ sends $((\Pi_{\mathtt{Comp}}, \mathtt{rid}), m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ to $\mathcal{F}_{\mathsf{Channel}}$ on behalf of P_i.
4. In $\Pi_{\mathtt{Compiler}}$, after $\mathcal{F}_{\mathsf{Channel}}$ has delivered the $(\Pi_{\mathtt{Comp}}, \mathtt{rid})$-th round message to an honest party, $\mathcal{S}$ learns from $\mathcal{F}_{\mathsf{Channel}}$ the set W' and updates $\mathcal{W} := \mathcal{W} \cup W'$.
   - If $|\mathcal{W}| > t_s - t_a$, $\mathcal{S}$ instructs $\mathcal{A}_{\mathtt{comp}}$ to set all messages sent from parties in $\mathcal{C}orr'$ to parties in $\mathcal{P}\backslash\mathcal{C}orr'$ to be all-0 strings in the following rounds of $\Pi_{\mathtt{Comp}}$. Then $\mathcal{S}$ breaks the loop.
   - If $\mathsf{mode} = \mathsf{asynch}$, for each party P_i in $\mathcal{W}$ that is not corrupted by $\mathcal{A}_{\mathtt{comp}}$ in $\Pi_{\mathtt{Comp}}$, $\mathcal{S}$ instructs $\mathcal{A}_{\mathtt{comp}}$ to corrupt P_i. Then $\mathcal{S}$ updates $\mathcal{C}orr' = \mathcal{C}orr \cup \mathcal{W}$.
5. In $\Pi_{\mathtt{Comp}}$, for each party $P_i \in \mathcal{W}$, $\mathcal{S}$ instructs $\mathcal{A}_{\mathtt{comp}}$ to set all P_i's messages sent/broadcast in the $\mathtt{rid}$-th round to be all-0 strings. For each party $P_i \in \mathcal{C}orr\backslash\mathcal{W}$, $\mathcal{S}$ instructs $\mathcal{A}_{\mathtt{comp}}$ to set P_i's messages to be the same as those received by $\mathcal{F}_{\mathsf{Channel}}$.

If $\mathcal{S}$ finishes the loop, $\mathcal{S}$ moves on to the reconstruction phase. If $\mathcal{S}$ breaks the loop, $\mathcal{S}$ moves to the fallback procedure.

In the reconstruction phase,

- If $\mathsf{mode} = \mathsf{synch}$, $\mathcal{S}$ sends (Input, x_i) for each $P_i \in \mathcal{C}orr \cap \mathcal{I}$ to $\mathcal{F}_{\mathsf{sfe}}$. Then $\mathcal{S}$ receives y from $\mathcal{F}_{\mathsf{sfe}}$.
- If $\mathsf{mode} = \mathsf{asynch}$, $\mathcal{S}$ sends (Input, x_i) for each $P_i \in \mathcal{C}orr \cap \mathcal{I}$ and sends $\mathcal{I}$ as $\mathsf{CoreSet}'$ to $\mathcal{F}_{\mathsf{sfe}}$. Then $\mathcal{S}$ receives y from $\mathcal{F}_{\mathsf{sfe}}^{L}$.

For each $i \in \{1, \ldots, n\}$, $\mathcal{S}$ randomly samples $[\![y_i]\!]$ given the secret $y_i = y$ and the shares of parties in $\mathcal{C}orr \cup \mathcal{W}$. Then $\mathcal{S}$ follows $\Pi_{\mathtt{Compiler}}$ in the reconstruction phase. Whenever an honest party P_i terminates with output y, $\mathcal{S}$ sends $(\mathsf{Deliver}, P_i)$ if $\mathsf{mode} = \mathsf{synch}$, and lets $\mathcal{F}_{\mathsf{sfe}}$ deliver the output to P_i if $\mathsf{mode} = \mathsf{asynch}$.

In the fallback procedure, $\mathcal{S}$ obtains $\mathcal{W}'$ of size $t_s - t_a$. To simulate the fallback procedure, let $\mathcal{A}_{\mathsf{asynch}}$ be the adversary for the underlying asynchronous MPC protocol $\Pi_{\mathtt{AMPC\text{-}GOD}}$ who just follows the instructions received from the environment and passes the messages received from honest parties in each round. Let $\mathcal{S}_{\mathsf{asynch}}$ be the simulator for $\mathcal{A}_{\mathsf{asynch}}$. $\mathcal{S}$ acts as $\mathcal{E}_{\mathsf{asynch}}$ and $\mathcal{F}_{\mathsf{AMPC}}$ and invokes $\mathcal{S}_{\mathsf{asynch}}$. $\mathcal{S}$ provides to all parties in $\mathcal{P}' := \mathcal{P}\backslash\mathcal{W}'$ all-0 strings as their inputs. For $\mathcal{F}_{\mathsf{AMPC}}$,

1. Define $\mathcal{C}orr' = \mathcal{C}orr\backslash\mathcal{W}'$.
2. Upon receiving an input v from a corrupted party $P_i \in \mathcal{C}orr'$, $\mathcal{S}$ records $x_i = v$ and notifies $\mathcal{S}_{\mathsf{asynch}}$. When an honest party $P_i \in \mathcal{P}'$ starts running $\Pi_{\mathtt{AMPC\text{-}GOD}}$ in $\Pi_{\mathtt{Compiler}}$, $\mathcal{S}$ notifies $\mathcal{S}_{\mathsf{asynch}}$ (that the input from P_i is received).
3. Upon receiving the core-set input $S \subset \mathcal{P}'$ of size at least $t_s + t_a + 1$ from $\mathcal{S}_{\mathsf{asynch}}$ for the first time, record it. If $\mathsf{mode} = \mathsf{synch}$, check that S contains all honest parties in $\mathcal{P}$. If not, $\mathcal{S}$ aborts.
4. Upon receiving inputs from all parties in the core-set, if $\mathsf{mode} = \mathsf{synch}$, $\mathcal{S}$ sends the inputs of corrupted parties in S to $\mathcal{F}_{\mathsf{sfe}}$. Otherwise, $\mathcal{S}$ sends S and the inputs of corrupted parties in S to $\mathcal{F}_{\mathsf{sfe}}$.
5. After receiving the function output y, $\mathcal{S}$ generates a request-based delayed output to send $P_i \in \mathcal{P}'$ the output y.

For $\mathcal{E}_{\mathsf{asynch}}$, $\mathcal{S}$ instructs $\mathcal{A}_{\mathsf{asynch}}$ to corrupt parties in $\mathcal{C}orr' := \mathcal{C}orr\backslash\mathcal{W}'$. Then

- Whenever $\mathcal{E}$ activates a party $P_i \in \mathcal{P}'$ in $\Pi_{\mathtt{Compiler}}$, $\mathcal{S}$ activates P_i in $\Pi_{\mathtt{AMPC\text{-}GOD}}$.
- For each message sent from $P_i \in \mathcal{C}orr'$ to $P_j \in \mathcal{P}'\backslash\mathcal{C}orr'$ in $\Pi_{\mathtt{Compiler}}$, $\mathcal{S}$ receives the message from $\mathcal{A}$ and instructs $\mathcal{A}_{\mathsf{asynch}}$ to let P_i sends the same message to P_j.
- For each message sent from $P_i \in \mathcal{P}'\backslash\mathcal{C}orr'$ to $P_j \in \mathcal{C}orr'$ in $\Pi_{\mathtt{AMPC\text{-}GOD}}$, $\mathcal{S}$ receives the message from $\mathcal{A}_{\mathsf{asynch}}$ and sends it to P_j on behalf of P_i in $\Pi_{\mathtt{Compiler}}$.

For each $P_i \in \mathcal{P}' \backslash \mathcal{C}orr'$, upon terminating $\Pi_{\texttt{AMPC-GOD}}$, $\mathcal{S}$ sends y to all parties on behalf of P_i and lets P_i terminates. For each $P_i \in \mathcal{W}' \backslash \mathcal{C}orr$, upon receiving $t_s + t_a + 1$ values from $\mathcal{P}'$, $\mathcal{S}$ lets P_i terminates. Whenever an honest party P_i terminates with output y, $\mathcal{S}$ sends $(\mathsf{Deliver}, P_i)$ if $\mathsf{mode} = \mathsf{synch}$, and lets $\mathcal{F}_{\mathsf{sfe}}$ deliver the output to P_i if $\mathsf{mode} = \mathsf{asynch}$.

Now we show that any computationally unbounded environment $\mathcal{E}$ cannot distinguish the real world and the ideal world.

$\mathbf{Hyb}_0$: In the initial hybrid, we consider the real world execution and define the output to be the output of $\mathcal{E}$.

$\mathbf{Hyb}_1$: In this hybrid, during the input phase for each dealer D, $\mathcal{S}$ starts an execution of $\Pi_{\texttt{Input}}(D)$ among parties in $\mathcal{P}$ and $\mathcal{S}$ performs as $\mathcal{E}_{\texttt{input}}$. The adversary $\mathcal{A}_{\texttt{input}}$ simply follows the instructions received from the environment and passes the messages received from honest parties. Initially, $\mathcal{S}$ instructs $\mathcal{A}_{\texttt{input}}$ to corrupt parties in $\mathcal{C}orr' := \mathcal{C}orr \cup \mathcal{W}$. If D is corrupted, $\mathcal{S}$ provides to D an all-0 string as his input. Otherwise, $\mathcal{S}$ provides to D his input of $\Pi_{\texttt{Compiler}}$.

Then from $\texttt{rid} = 1$ to R_I,

1. $\mathcal{S}$ receives from $\mathcal{A}_{\texttt{input}}$ P2P messages and broadcast messages that parties in $\mathcal{P} \backslash \mathcal{C}orr'$ send to parties in $\mathcal{C}orr'$.
2. In $\Pi_{\texttt{Compiler}}$, for each honest party $P_i \notin \mathcal{W}$, after P_i enters the $\texttt{rid}$-th round, $\mathcal{S}$ prepares $(m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ as follows: $\mathcal{S}$ sets $m_0^{(i)}$ as the broadcast message of P_i received from $\mathcal{A}_{\texttt{input}}$. For each $P_j \in \mathcal{C}orr'$, $\mathcal{S}$ sets $m_j^{(i)}$ as the P2P message sent from P_i to P_j received from $\mathcal{A}_{\texttt{input}}$. For each party $P_j \in \mathcal{P} \backslash \mathcal{C}orr'$, $\mathcal{S}$ sets $m_j^{(i)}$ to be an all-0 string. Then $\mathcal{S}$ sends $((\Pi_{\texttt{Input}}, D, \texttt{rid}), m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ to $\mathcal{F}_{\mathsf{Channel}}$ on behalf of P_i.
3. In $\Pi_{\texttt{Compiler}}$, for each honest party $P_i \in \mathcal{W}$, after P_i enters the $\texttt{rid}$-th round, $\mathcal{S}$ sets $(m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ as all-0 strings. Then $\mathcal{S}$ sends $((\Pi_{\texttt{Input}}, D, \texttt{rid}), m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ to $\mathcal{F}_{\mathsf{Channel}}$ on behalf of P_i.
4. In $\Pi_{\texttt{Compiler}}$, after $\mathcal{F}_{\mathsf{Channel}}$ has delivered the $(\Pi_{\texttt{Input}}, D, \texttt{rid})$-th round message to an honest party, $\mathcal{S}$ learns from $\mathcal{F}_{\mathsf{Channel}}$ the set W' and updates $\mathcal{W} := \mathcal{W} \cup W'$.
   - If $D \in \mathcal{W}$ or $|\mathcal{W}| > t_s - t_a$, $\mathcal{S}$ instructs $\mathcal{A}_{\texttt{input}}$ to set all messages sent from parties in $\mathcal{C}orr'$ to parties in $\mathcal{P} \backslash \mathcal{C}orr'$ to be all-0 strings in the following rounds of $\Pi_{\texttt{Input}}$. Then $\mathcal{S}$ breaks the loop.
   - If $\mathsf{mode} = \mathsf{asynch}$, for each party P_i in $\mathcal{W}$ that is not corrupted by $\mathcal{A}_{\texttt{input}}$ in $\Pi_{\texttt{Input}}$, $\mathcal{S}$ instructs $\mathcal{A}_{\texttt{input}}$ to corrupt P_i. Then $\mathcal{S}$ updates $\mathcal{C}orr' = \mathcal{C}orr \cup \mathcal{W}$.
5. In $\Pi_{\texttt{Input}}$, for each party $P_i \in \mathcal{W}$, $\mathcal{S}$ instructs $\mathcal{A}_{\texttt{input}}$ to set all P_i's messages sent/broadcast in the $\texttt{rid}$-th round to be all-0 strings. For each party $P_i \in \mathcal{C}orr \backslash \mathcal{W}$, $\mathcal{S}$ instructs $\mathcal{A}_{\texttt{input}}$ to set P_i's messages to be the same as those received by $\mathcal{F}_{\mathsf{Channel}}$.

If $\mathcal{S}$ finishes the loop, $\mathcal{S}$ adds D in $\mathcal{I}$ and moves on to the next dealer. If $\mathcal{S}$ breaks the loop due to $D \in \mathcal{W}$, $\mathcal{S}$ moves on to the next dealer. If $\mathcal{S}$ breaks the loop due to $|\mathcal{W}| > t_s - t_a$, $\mathcal{S}$ moves to the fallback procedure.

We show that $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ are identically distributed. Note that the only difference is that for each honest party P_i, we use the messages generated by P_i in $\Pi_{\texttt{Input}}(D)$ for $\Pi_{\texttt{Compiler}}$. Note that in each round, every honest party not in $\mathcal{C}orr'$ generates his messages following $\Pi_{\texttt{Input}}(D)$ using his local input, randomness, and all previous received messages, which is also the case in $\Pi_{\texttt{Compiler}}$. Also in each round, $\mathcal{S}$ will crash honest parties in $\Pi_{\texttt{Input}}(D)$ in the same way as that in $\Pi_{\texttt{Compiler}}$. Thus, by induction, the messages exchanged in $\Pi_{\texttt{Input}}(D)$ in $\mathbf{Hyb}_1$ and those in $\Pi_{\texttt{Compiler}}$ in $\mathbf{Hyb}_0$ are identically distributed.

$\mathbf{Hyb}_2$: By the security of $\Pi_{\texttt{Input}}(D)$, after replacing $\mathcal{A}_{\texttt{input}}$ and honest parties by $\mathcal{S}_{\texttt{input}}$ and $\mathcal{F}_{\mathsf{Input}}(D)$, the distribution is statistically close to that in $\mathbf{Hyb}_1$.

$\mathbf{Hyb}_3$: In this hybrid, during the computation phase, $\mathcal{S}$ starts an execution of $\Pi_{\texttt{Comp}}$ among parties in $\mathcal{P}$ and $\mathcal{S}$ performs as $\mathcal{E}_{\texttt{comp}}$. The adversary $\mathcal{A}_{\texttt{comp}}$ simply follows the instructions received from the environment and passes the messages received from honest parties. Initially, $\mathcal{S}$ instructs $\mathcal{A}_{\texttt{comp}}$ to corrupt parties in $\mathcal{C}orr' := \mathcal{C}orr \cup \mathcal{W}$. For each corrupted party, $\mathcal{S}$ provides $\mathcal{I}$ and all-0 strings as his shares of $\{[\![x_i]\!]\}_{P_i \in \mathcal{I}}$. For each honest party, $\mathcal{S}$ provides $\mathcal{I}$ and the actual share generated by $\mathcal{F}_{\mathsf{Input}}(D)$ for each $D \in \mathcal{I}$.

Then from $\texttt{rid} = 1$ to R_C,

1. $\mathcal{S}$ receives from $\mathcal{A}_{\texttt{comp}}$ P2P messages and broadcast messages that parties in $\mathcal{P} \backslash \mathcal{C}orr'$ send to parties in $\mathcal{C}orr'$.

2. In $\Pi_{\texttt{Compiler}}$, for each honest party $P_i \notin \mathcal{W}$, after P_i enters the $\texttt{rid}$-th round, $\mathcal{S}$ prepares $(m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ as follows: $\mathcal{S}$ sets $m_0^{(i)}$ as the broadcast message of P_i received from $\mathcal{A}_{\texttt{comp}}$. For each $P_j \in \mathcal{C}orr'$, $\mathcal{S}$ sets $m_j^{(i)}$ as the P2P message sent from P_i to P_j received from $\mathcal{A}_{\texttt{comp}}$. For each party $P_j \in \mathcal{P}\backslash\mathcal{C}orr'$, $\mathcal{S}$ sets $m_j^{(i)}$ to be an all-0 string. Then $\mathcal{S}$ sends $((\Pi_{\texttt{Comp}}, \texttt{rid}), m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ to $\mathcal{F}_{\mathsf{Channel}}$ on behalf of P_i.
3. In $\Pi_{\texttt{Compiler}}$, for each honest party $P_i \in \mathcal{W}$, after P_i enters the $\texttt{rid}$-th round, $\mathcal{S}$ sets $(m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ as all-0 strings. Then $\mathcal{S}$ sends $((\Pi_{\texttt{Comp}}, \texttt{rid}), m_0^{(i)}, m_1^{(i)}, \ldots, m_n^{(i)})$ to $\mathcal{F}_{\mathsf{Channel}}$ on behalf of P_i.
4. In $\Pi_{\texttt{Compiler}}$, after $\mathcal{F}_{\mathsf{Channel}}$ has delivered the $(\Pi_{\texttt{Comp}}, \texttt{rid})$-th round message to an honest party, $\mathcal{S}$ learns from $\mathcal{F}_{\mathsf{Channel}}$ the set W' and updates $\mathcal{W} := \mathcal{W} \cup W'$.
   - If $|\mathcal{W}| > t_s - t_a$, $\mathcal{S}$ instructs $\mathcal{A}_{\texttt{comp}}$ to set all messages sent from parties in $\mathcal{C}orr'$ to parties in $\mathcal{P}\backslash\mathcal{C}orr'$ to be all-0 strings in the following rounds of $\Pi_{\texttt{Comp}}$. Then $\mathcal{S}$ breaks the loop.
   - If $\mathsf{mode} = \mathsf{asynch}$, for each party P_i in $\mathcal{W}$ that is not corrupted by $\mathcal{A}_{\texttt{comp}}$ in $\Pi_{\texttt{Comp}}$, $\mathcal{S}$ instructs $\mathcal{A}_{\texttt{comp}}$ to corrupt P_i. Then $\mathcal{S}$ updates $\mathcal{C}orr' = \mathcal{C}orr \cup \mathcal{W}$.
5. In $\Pi_{\texttt{Comp}}$, for each party $P_i \in \mathcal{W}$, $\mathcal{S}$ instructs $\mathcal{A}_{\texttt{comp}}$ to set all P_i's messages sent/broadcast in the $\texttt{rid}$-th round to be all-0 strings. For each party $P_i \in \mathcal{C}orr\backslash\mathcal{W}$, $\mathcal{S}$ instructs $\mathcal{A}_{\texttt{comp}}$ to set P_i's messages to be the same as those received by $\mathcal{F}_{\mathsf{Channel}}$.

If $\mathcal{S}$ finishes the loop, $\mathcal{S}$ moves on to the reconstruction phase. If $\mathcal{S}$ breaks the loop, $\mathcal{S}$ moves to the fallback procedure.

Following the same argument in $\mathbf{Hyb}_1$, $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ are identically distributed.

$\mathbf{Hyb}_4$: By the security of $\Pi_{\texttt{Comp}}$, after replacing $\mathcal{A}_{\texttt{comp}}$ and honest parties by $\mathcal{S}_{\texttt{comp}}$ and $\mathcal{F}_{\mathsf{Comp}}$, the distribution is statistically close to that in $\mathbf{Hyb}_3$.

$\mathbf{Hyb}_5$: In this hybrid, in the fallback procedure, $\mathcal{S}$ starts an execution of $\Pi_{\texttt{AMPC-GOD}}$ among parties in $\mathcal{P}' := \mathcal{P}\backslash\mathcal{W}'$ and $\mathcal{S}$ performs as $\mathcal{E}_{\mathsf{asynch}}$. The adversary $\mathcal{A}_{\mathsf{asynch}}$ simply follows the instructions received from the environment and passes the messages received from honest parties. Initially, $\mathcal{S}$ instructs $\mathcal{A}_{\mathsf{asynch}}$ to corrupt parties in $\mathcal{C}orr' := \mathcal{C}orr\backslash\mathcal{W}'$. For each corrupted party, $\mathcal{S}$ provides an all-0 strings as his input. For each honest party, $\mathcal{S}$ provides his actual input of $\Pi_{\texttt{Compiler}}$.

Then

- Whenever $\mathcal{E}$ activates a party $P_i \in \mathcal{P}'$ in $\Pi_{\texttt{Compiler}}$, $\mathcal{S}$ activates P_i in $\Pi_{\texttt{AMPC-GOD}}$.
- For each message sent from $P_i \in \mathcal{C}orr'$ to $P_j \in \mathcal{P}'\backslash\mathcal{C}orr'$ in $\Pi_{\texttt{Compiler}}$, $\mathcal{S}$ receives the message from $\mathcal{A}$ and instructs $\mathcal{A}_{\mathsf{asynch}}$ to let P_i sends the same message to P_j.
- For each message sent from $P_i \in \mathcal{P}'\backslash\mathcal{C}orr'$ to $P_j \in \mathcal{C}orr'$ in $\Pi_{\texttt{AMPC-GOD}}$, $\mathcal{S}$ receives the message from $\mathcal{A}_{\mathsf{asynch}}$ and sends it to P_j on behalf of P_i in $\Pi_{\texttt{Compiler}}$.

For each $P_i \in \mathcal{P}'\backslash\mathcal{C}orr'$, upon terminating $\Pi_{\texttt{AMPC-GOD}}$, $\mathcal{S}$ sends y to all parties on behalf of P_i and lets P_i terminates.

Following the same argument in $\mathbf{Hyb}_1$, $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ are identically distributed.

$\mathbf{Hyb}_6$: By the security of $\Pi_{\texttt{AMPC-GOD}}$, after replacing $\mathcal{A}_{\mathsf{asynch}}$ and honest parties by $\mathcal{S}_{\mathsf{asynch}}$ and $\mathcal{F}_{\mathsf{AMPC}}$, the distribution is statistically close to that in $\mathbf{Hyb}_5$.

$\mathbf{Hyb}_7$: In this hybrid, $\mathcal{F}_{\mathsf{AMPC}}$ is simulated by $\mathcal{S}$ as described in the construction of the simulator and $\mathcal{S}$ when acting as $\mathcal{E}_{\mathsf{asynch}}$ no longer provides to each honest party in $\mathcal{P}'$ his actual input. It is sufficient to argue that if $\mathsf{mode} = \mathsf{synch}$, $\mathsf{core-set}$ contains all honest parties. By the property of $\Pi_{\texttt{AMPC-GOD}}$, all honest parties in $\mathcal{P}'$ are in $\mathsf{core-set}$. Since when $\mathsf{mode} = \mathsf{synch}$, by $\mathcal{F}_{\mathsf{Channel}}$, $\mathcal{W}'$ only contain corrupted parties. Thus all honest parties in $\mathcal{P}$ are also in $\mathcal{P}'$. Therefore, $\mathbf{Hyb}_7$ and $\mathbf{Hyb}_6$ are identically distributed.

$\mathbf{Hyb}_8$: In this hybrid, in the reconstruction phase, $\mathcal{S}$ resamples each $[\![y_i]\!]$ by using the shares of parties in $\mathcal{C}orr \cup \mathcal{W}$ and y_i. By the efficient resampling property of the robust secret sharing scheme, $\mathbf{Hyb}_8$ and $\mathbf{Hyb}_7$ are statistically close.

$\mathbf{Hyb}_9$: In this hybrid, in the reconstruction phase,

- If $\mathsf{mode} = \mathsf{synch}$, $\mathcal{S}$ sends (Input, x_i) for each $P_i \in \mathcal{C}orr \cap \mathcal{I}$ to $\mathcal{F}_{\mathsf{sfe}}$. Then $\mathcal{S}$ receives y from $\mathcal{F}_{\mathsf{sfe}}$.
- If $\mathsf{mode} = \mathsf{asynch}$, $\mathcal{S}$ sends (Input, x_i) for each $P_i \in \mathcal{C}orr \cap \mathcal{I}$ and sends $\mathcal{I}$ as $\mathsf{CoreSet}'$ to $\mathcal{F}_{\mathsf{sfe}}$. Then $\mathcal{S}$ receives y from $\mathcal{F}_{\mathsf{sfe}}^{L}$.

We argue that $\mathbf{Hyb}_9$ and $\mathbf{Hyb}_8$ are identically distributed.

It is sufficient to argue that when $\mathsf{mode} = \mathsf{synch}$, $\mathcal{I}$ contains all honest parties, and when $\mathsf{mode} = \mathsf{asynch}$, $\mathcal{I}$ is of size at least $t_s + t_a + 1$. For the former statement, note that $\mathcal{W}$ only contain corrupted parties. When $|\mathcal{W}| \leq t_s - t_a$ at the end of the input phase, all $\Pi_{\mathtt{Input}}(D)$ have been successfully finished. Thus all honest parties are in $\mathcal{I}$. For the latter state, note that $|\mathcal{I}| \geq n - |\mathcal{W}| \geq n - (t_s - t_a) \geq t_s + t_a + 1$.

$\mathbf{Hyb}_{10}$: In this hybrid, $\mathcal{F}_{\mathsf{Comp}}$ is simulated by $\mathcal{S}$ as described in the construction of the simulator and $\mathcal{S}$ when acting as $\mathcal{E}_{\mathtt{comp}}$ no longer provides to each honest party not in $\mathcal{W}$ his actual shares of $\{[\![x_i]\!]\}_{P_i \in \mathcal{I}}$. By the adaptive secrecy property of the robust secret sharing scheme, $\mathbf{Hyb}_{10}$ and $\mathbf{Hyb}_9$ are statistically close.

$\mathbf{Hyb}_{11}$: In this hybrid, $\mathcal{F}_{\mathsf{Input}}(D)$ is simulated by $\mathcal{S}$ as described in the construction of the simulator and $\mathcal{S}$ when acting as $\mathcal{E}_{\mathtt{input}}$ no longer provides to an honest D his actual input. By the adaptive secrecy property of the robust secret sharing scheme, $\mathbf{Hyb}_{11}$ and $\mathbf{Hyb}_{10}$ are statistically close.

Note that $\mathbf{Hyb}_{11}$ is the ideal world. This concludes the proof.