Perfectly Secure Network-Agnostic MPC Comes for Free

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

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

Abstract. Secure multiparty computation (MPC) allows a set of parties to jointly compute a function while keeping their inputs private. Classical MPC protocols assume either a synchronous or an asynchronous network. Synchronous protocols tolerate more corrupted parties but rely on a timing bound, while asynchronous protocols make no timing assumptions but handle fewer corruptions.

The network-agnostic model aims to combine the advantages of both. It requires security without knowing in advance whether the network is synchronous or asynchronous, guaranteeing resilience against up to t_s corruptions in the synchronous case and t_a corruptions in the asynchronous case. The optimal corruption threshold for perfect security has been established as $n = 2\max(t_s, t_a) + \max(2t_a, t_s) + 1$, but prior work either falls short of this threshold or requires exponential local computation.

In this work, we present the first perfectly secure network-agnostic MPC protocol with polynomial communication and computation complexity under the optimal threshold. Our protocol achieves expected communication complexity $\mathcal{O}((|C|n + (D + C_I)n^2 + n^6)\log n)$ bits for a circuit of size $|C|$ over a finite field $\mathbb{F}$ of size $\mathcal{O}(n)$, depth D, and input size C_I.

Our main technical contribution is a compiler that generates Beaver triples in the network-agnostic setting using synchronous and asynchronous triple-generation protocols in a black-box way. Beyond the cost of the underlying protocols, it only requires $\mathcal{O}(n^2)$ instances of network-agnostic Byzantine agreement.

1 Introduction

Secure multiparty computation (MPC) [Yao82, GMW87, BGW88, CCD88, RB89] enables a set of n parties to jointly compute a function on their private inputs while ensuring that any t corrupted parties learn nothing beyond what can be inferred from their own inputs and outputs.

Traditionally, the design of MPC protocols assumes a fixed communication model: either *synchronous* or *asynchronous*. In the synchronous model, parties proceed in lock-step rounds with synchronized clocks, and every message is guaranteed to be delivered within a known time bound Δ. This assumption enables strong security guarantees: statistically secure MPC can be achieved for $t < n/2$ given broadcast [RB89, DN07, BSFO12, GSZ20], and perfectly secure MPC is feasible for up to $t < n/3$ corrupted parties with guaranteed output delivery [BGW88, BH08]. The drawback, however, is brittleness: once messages are delayed beyond Δ, the protocol may completely lose security.

In contrast, the asynchronous model makes no assumptions on delivery time—
messages may arrive after any finite delay, but honest parties' messages are guaranteed to eventually be delivered. This makes asynchronous protocols robust to unpredictable networks, but comes with a cost in corruption tolerance: statistically secure asynchronous MPC requires $t < n/3$ [BKR94, CGHZ16, HNP05, Coh16, CHLZ21, GLZS24, BJK+25b, BJK+25a], and perfect security is possible only up to $t < n/4$ corruptions [BCG93, AAPP24]. In other words, synchronous protocols achieve higher resilience but rely on timing assumptions, while asynchronous protocols drop timing assumptions but tolerate fewer corruptions.

This trade-off naturally raises the question: can we design protocols that remain secure without knowing in advance whether the network is synchronous or asynchronous, and thereby inherit the best possible guarantees in either case? This is precisely the goal of the *network-agnostic model*. In this setting, parties are unaware of the underlying network type, yet the protocol guarantees security against up to t_s corruptions if the network is synchronous and up to t_a corruptions if the network is asynchronous. This model has received

increased attention in recent years and been considered in the study of multiple primitives including agreement [BKL19, DHLZ21, BKL21, ABKL22, GLZW22, GLZW23], secure-communication [DLZ23], distributed key generation [BCLZL23], and MPC [BZL20, DHLZ21, BCLZL23, ACC22, AC23, BCV24, PP24].

The study of perfectly secure network-agnostic MPC was initiated by [ACC22], who achieved amortized $\mathcal{O}(n^5/t_a \log n)$ bits communication per multiplication gate with an additional $\mathcal{O}(n^7 \log n)$ additive overhead, under the corruption threshold $n = 3t_s + t_a + 1$. Later, [PP24] established the exact optimal threshold for perfectly secure network-agnostic MPC as $n = 2\max(t_s, t_a) + \max(2t_a, t_s) + 1$, and presented a matching feasibility result. Their construction, however, relies on all parties agreeing on a clique of size $n - t_s$, which requires exponential local computation complexity. Consequently, no existing protocol achieves both polynomial communication and computation under the optimal corruption threshold. This motivates our work: constructing the first perfectly secure network-agnostic MPC with polynomial efficiency. At the same time, we are curious whether one can leverage existing synchronous and asynchronous MPC techniques and build a generic compiler for the network-agnostic setting, rather than designing protocols from scratch.

1.1 Our Contributions

In this work, we study the design of perfectly secure network-agnostic MPC with both polynomial communication and computation complexity, under the optimal corruption threshold $n = 2\max(t_s, t_a) + \max(2t_a, t_s) + 1$ [PP24]. Our primary focus is on the amortized communication complexity required to evaluate each gate in the circuit.

For perfect security, existing works have already achieved amortized linear communication per multiplication gate in both the synchronous setting [BH08, GLS19, AAPP23] and the asynchronous setting [AAPP24, AAPS25], while keeping the computation complexity polynomial in the number of parties. This sharp contrast highlights a natural open question:

Can we also obtain a perfectly secure network-agnostic MPC with amortized linear communication and polynomial computation complexity?

We answer this question positively and obtain the following result.

Theorem 1. *Let n, t_s, t_a be positive integers such that $n = 2\max(t_s, t_a) + \max(2t_a, t_s) + 1$. Let f be an n-ary function, there is a perfectly 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 $|C|$ and circuit depth D, let C_I denote the input size, the achieved communication complexity is $\mathcal{O}((|C|n + (D + C_I)n^2 + n^6) \log n)$ bits in expectation.*

Note that under the optimal corruption threshold $n = 2\max(t_s, t_a) + \max(2t_a, t_s) + 1$, the case $t_s \leq t_a$ reduces to $n = 4t_a + 1$. In this regime, one can simply run any perfectly secure asynchronous MPC protocol tolerating t_a corruptions. Hence, in the remainder of this work, we focus on the more challenging case $t_a < t_s$ and present our construction for this setting.

Main Technical Result. Following [PP24], we divide the construction of network-agnostic MPC into two phases: the offline phase and the online phase. In the offline phase, the parties prepare random Beaver triples [Bea92], which are then used to evaluate the circuit in the online phase. Unlike prior works [ACC22, PP24], our approach does not construct network-agnostic MPC by first building a network-agnostic secret-sharing scheme. Instead, we build the network-agnostic triple generation protocol by using a synchronous and asynchronous triple generation protocol in a black-box way. This yields the following theorem.

Theorem 2. *Let n, t_s, t_a be positive integers such that $n = 2\max(t_s, t_a) + \max(2t_a, t_s) + 1$. There exists a perfectly secure network-agnostic triple generation protocol that generates N random Beaver triples with communication complexity $\mathcal{O}((Nn + n^6) \log n)$ bits.*

2 Technical Overview

In the following, let n denote the number of parties, t_s denote the synchronous corruption threshold, and t_a denote the asynchronous corruption threshold. We also assume that $t_a < t_s$ and set $n = 2t_s + \max(2t_a, t_s) + 1$.

In this section, we introduce the high-level idea of our network-agnostic triple generation protocol. Concretely, we construct a compiler that combines synchronous and asynchronous triple generation protocols to obtain a network-agnostic one. Recent works [DME24, JLZPS26] also propose compilers for constructing network-agnostic Byzantine agreement and MPC. However, their constructions focus on statistical security and rely on digital signatures, and therefore cannot be applied in the perfectly secure setting.

Throughout this subsection, when we say that a party broadcasts a message, we mean that it invokes an instance of network-agnostic reliable broadcast protocol with that message as input.

2.1 Safely Execute the Synchronous Protocol

We first attempt to let parties invoke the synchronous protocol in the network-agnostic setting. We note that the security of the synchronous protocol relies on two guarantees from the synchronous network: (1) all (honest) parties will receive their messages (from honest parties) within a fixed time bound in each round, and (2) all (honest) parties will proceed to the next round at the same time. However, if the network is asynchronous, we may face the following security and termination issues.

- An honest party A may fail to receive a message from another party B within the expected time bound. However, A cannot distinguish whether this is due to network asynchrony causing message delay, or because a corrupted B drops the message. As a result, A cannot determine whether it has already received the required messages from all honest parties in the current round, and thus may wait indefinitely.
- Even if an honest party receives all messages in a given round, it cannot guarantee that all other honest parties have done so. Since the security of a synchronous protocol may require that honest parties progress round-by-round in a coordinated manner, this prevents the party from safely moving to the next round.

To let parties make progress while maintaining the security, our idea is to design a protocol $\Pi_{\texttt{Channel}}$ that helps parties to deliver their messages in each round of the synchronous protocol. If all parties acknowledge that all messages are delivered to each other, then they can safely proceed to the next round. Otherwise, if several parties' messages are delayed and cause the protocol stall, $\Pi_{\texttt{Channel}}$ would help to identify parties that are responsible for this stall, and let all parties abort the protocol. With the help of $\Pi_{\texttt{Channel}}$, parties can safely execute a synchronous protocol in the network-agnostic setting, although they cannot guarantee the generation of triples so far. We first introduce the construction of $\Pi_{\texttt{Channel}}$ as follows, then show how to utilize it in our compiler for triple generation in the next subsection.

Realization of $\Pi_{\texttt{Channel}}$. For each ordered pair (A, B), we let the receiver A broadcast a confirmation message $(\texttt{Confirm}, A, B)$ if it received B's message within the prescribed time bounds, and a conflict message $(\texttt{Conflict}, A, B)$ otherwise. To prevent from waiting indefinitely due to a corrupted receiver A never broadcasting either a confirmation or a conflict message, parties invoke a network-agnostic Byzantine agreement (BA) protocol with input 1 if they have received a broadcast from A regarding whether A has received B's message within the prescribed time bound, and 0 otherwise.

- If BA outputs 1, then at least one honest party has received A's broadcast. Honest parties wait until they receive either $(\texttt{Confirm}, A, B)$ or $(\texttt{Conflict}, A, B)$ from A.
- If BA outputs 0, then not all honest parties received A's broadcast within the time bound. In this case, parties can know that if the network is synchronous, then A is a corrupted party. Then they treat $(\texttt{Conflict}, A, B)$ as the agreed outcome for this pair.

Note that with the help of the time bound, all honest parties eventually will have an input for each BA, thus guaranteeing the termination of these BA protocols. After parties reach an agreement on a confirmation or conflict outcome for every pair, if all outcomes are confirmations, then all required messages for the current

round have been received, and parties can safely advance to the next round of the synchronous protocol. Otherwise, if at least one outcome is a conflict, all parties can agree on the identity of parties (A, B) that are responsible for this failure. Refer to Section 4.2 for the detailed construction.

A Subtle Issue. In the synchronous case, when we say an honest party A can receive messages from another honest party B within prescribed time bound, we also require that both A and B should enter this round at a similar time. Since the time to enter the next round depends on the termination of BA or broadcast protocol in previous round, we need to ensure that all honest parties should terminate the BA or broadcast protocol at a similar time when the network is synchronous. This issue can be addressed with a modified version of Bracha's reliable broadcast protocol [Bra84], and we show the details in Section 4.1.

2.2 High-level Idea of the Compiler

With the help of $\Pi_{\texttt{Channel}}$, we guarantee that once parties fail to make the progress in the synchronous protocol, they can identify a pair of parties (A, B) that are responsible for this failure. Then our first observation is that in a synchronous case, every honest party will receive all honest parties' messages within the time bound. Consequently, for a pair of parties (A, B) that are responsible for the failure of the synchronous protocol, this ensures the following:

- If the network is synchronous, then at least one of A or B must be corrupted.
- If the network is asynchronous, it is possible that both A and B are honest.

Starting from this observation, we further note that if we directly remove both A and B upon failure, then among the remaining $n-2$ parties there are at most $t_s - 1$ corrupted parties in the synchronous case and at most t_a corrupted parties in the asynchronous case. Moreover, the relation $n-2 = 2(t_s-1)+\max(2t_a, t_s-1)+1$ still holds. Therefore, the optimal corruption threshold is preserved for the remaining parties, and they can rerun the synchronous protocol.

If the protocol fails again, the parties eliminate another pair and repeat the procedure. Note that the number of corrupted parties will decrease by 1 in the synchronous case but 0 in the asynchronous case, so we will only repeats this elimination procedure for $t_s - t_a$ times to reduce the number of corrupt parties to t_a in both network scenarios. We note that in this case, since the number of remaining parties is at least

$$n - 2(t_s - t_a) = 2t_a + \max(2t_a, t_s) + 1 = \max(4t_a, 2t_a + t_s) + 1 \geq 4t_a + 1,$$

the remaining parties can safely switch to a perfectly secure asynchronous triple generation protocol that tolerates up to t_a corrupted parties.

To summarize, now parties can first attempt to safely invoke the synchronous protocol (with the help of $\Pi_{\texttt{Channel}}$) to generate triples. If they fail due to the delayed messages of several parties, they would directly eliminate these parties and let the remaining parties rerun the synchronous protocol. Upon eliminating a sufficient number of these parties, the remaining parties can safely switch to execute the asynchronous protocol to generate triples.

Further Optimization. If the remaining parties generate the full set of triples after each elimination, the procedure may repeat $t_s - t_a = \mathcal{O}(n)$ times, leading to at least quadratic amortized communication per triple. To achieve linear amortized cost, we adopt the party elimination framework of [HMP00]. Specifically, we divide triple generation into n segments, where each segment produces $1/n$ fraction of the triples.

However, a subtle issue arises: triples generated in different segments may have different degrees. Let $t_k = t_s - k$ denote the corruption threshold after eliminating $2k$ parties, and let $\mathcal{P}_k$ denote the remaining $n - 2k$ parties. If an arbitrary synchronous protocol is used, the parties in $\mathcal{P}_k$ generate triples of the form $([a]_{t_k}, [b]_{t_k}, [c]_{t_k})$, where each sharing has degree t_k. Thus, triples produced in different segments may have different degrees, which complicates parallel online evaluation and introduces additional overhead.

To resolve this, we use the protocol of [BH08]. Let n' denote the current number of parties, t' the number of corrupted parties, and d the desired degree of the triples. The protocol of [BH08] guarantees that the parties can generate degree-d triples as long as $d + 2t' < n'$.

In our setting, after k elimination steps, we have $n' = n - 2k$ and $t' = t_s - k$. We set the target degree to $d = t_s$. The condition holds as long as $k \leq t_s - t_a$:

$$d + 2t' = t_s + 2(t_s - k) < 2t_s + \max(2t_a, t_s) + 1 - 2k = n'.$$

Therefore, throughout the execution, the parties can consistently generate degree-t_s triples when invoking the synchronous triple generation protocols [BH08].

Remark 1. The protocol in [BH08] we used actually only achieves security with identifiable abort. That means when all parties terminate, they either output degree-t_s triples or agree on a pair of parties who are responsible for the failure of the protocol. The protocol ensures that at least one of these two parties is corrupted. Following a similar argument in Section 2.2, all parties can directly eliminate this pair of parties and let the remaining parties rerun the current segment.

Putting It All Together. We summarize the compiler as follows. All parties first try to invoke the synchronous protocol to generate degree-t_s triples. During this procedure, triple generation is divided into n segments. In each segment, the parties proceed as follows.

1. All parties execute a synchronous triple generation protocol with identifiable abort [BH08]. Messages in each round are delivered via $\Pi_{\texttt{Channel}}$.
2. Upon terminating $\Pi_{\texttt{Channel}}$, the parties either agree on a pair of parties to be eliminated or proceed to the next round. At the end of the segment:
   - If valid degree-t_s triples are produced, the parties proceed to the next segment.
   - Otherwise, the identified parties are eliminated and the segment is restarted.

During the execution of these n segments, if a total of $2(t_s - t_a)$ parties have been eliminated, all intermediate results from the synchronous phase are discarded. The remaining $4t_a + 1$ parties then invoke an asynchronous triple generation protocol to produce degree-t_a triples. Refer to Section 5.1 for the detailed construction of the compiler.

2.3 Online Error Correction Still Works

During the circuit evaluation, with the help of triples, parties only need to do public reconstruction. We use a modified version of the reconstruction protocol introduced in [DN07], and refer the reader to Section 5.2 for more details. It remains to show that the online error-correction procedure works in our setting to guarantee the correctness of the public reconstruction.

Suppose that the parties have terminated the offline phase and agreed on a subset of parties $\mathcal{P}'$ that hold shares of the generated triples. Let $|\mathcal{P}'| = n - 2k$ for some $k \in [0, t_s - t_a]$. If $k = t_s - t_a$, then the parties in $\mathcal{P}'$ hold degree-t_a triples and there are at most t_a corrupted parties. Otherwise, if $k < t_s - t_a$, the parties hold degree-t_s triples and the number of corrupted parties is at most $t_s - k$.

When $|\mathcal{P}'| = 4t_a + 1$ and $t = t_a$, correctness follows directly. We now consider the case $|\mathcal{P}'| = n_k = n - 2k$ for $k \in [0, t_s - t_a)$ and $t = t_s$. In this case, we have

$$\begin{aligned}\frac{n_k - t - 1}{2} = \frac{n - 2k - t_s - 1}{2} &= \frac{t_s + \max(2t_a, t_s) - 2k}{2} \\ &= \frac{1}{2}\max(2t_a + t_s - 2k,\ 2t_s - 2k) \geq t_s - k.\end{aligned}$$

Therefore, the reconstruction procedure can tolerate up to $t_s - k$ incorrect shares. Since there are at most $t_s - k$ corrupted parties in $\mathcal{P}'$, all incorrect values can be corrected, and the online phase succeeds. The same argument guarantees correctness of output reconstruction.

3 Preliminary

3.1 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] 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.

3.2 Ideal World

We consider the setting of multi-party computation with guaranteed output delivery (GOD). In this setting, we model the ideal MPC functionality $\mathcal{F}_{\mathsf{sfe}}$ defined below with a flag mode, indicating whether the setting is synchronous or asynchronous. When the network is synchronous, the functionality will take into account all inputs from honest parties for the computation. 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]. In both cases, the outputs are eventually delivered.

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.
For each party P_i, initialize an input value $x^{(i)} = \perp$ and output value $y^{(i)} = \perp$.

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]. Otherwise, $\mathcal{F}_{\mathsf{sfe}}$ initializes $\mathsf{CoreSet}$ to be the set $\mathcal{H}$ of all honest parties.

Computation

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: If $x^{(i)}$ has been set to a value different from $\perp$ for every honest $P_i \in \mathsf{CoreSet}$, then compute $y = f(x^{(1)}, \ldots, x^{(n)})$ (set $x^{(i)} = \perp$ for each $P_i \notin \mathsf{CoreSet}$) and generate a request-based delayed output $y^{(i)} = y$ for every $P_i \in \mathcal{P}$.
3: Upon receiving an input $\mathsf{CoreSet}'$ from $\mathcal{S}$ and the output has not been delivered to any party, verify that $\mathsf{CoreSet}'$ is a subset of $\mathcal{P}$ of size at least $n - t_s$ (and if $\mathsf{mode} = \mathsf{synch}$, also check that $\mathsf{CoreSet}'$ contains all honest parties), else ignore the message. Then record $\mathsf{CoreSet} = \mathsf{CoreSet}'$. Finally, $\mathcal{F}_{\mathsf{sfe}}$ re-executes Step 2 and updates all generated request-based delayed output according to the new $\mathsf{CoreSet}$.

[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.

3.3 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.

3.4 Network-Agnostic Reliable Broadcast

In a reliable broadcast (RBC) protocol, a sender P_s takes an L-bit message $m \in \{0,1\}^L$ as input. The RBC protocol allows the sender to consistently distribute the message m among a set of parties. We recall the property-based definition of the network-agnostic RBC protocol in [ACC22, PP24] as follows.

Definition 1 ([ACC22, PP24]). *Let t be the corruption threshold, $L \in \mathbb{N}$ be the input length, and $\mathcal{P}$ be the party set. Let Π be a protocol executed by parties in $\mathcal{P}$ with up to t corruptions, a sender P_s takes a value $m \in \{0,1\}^L$ as input, and all parties terminate after getting the output $m' \in \{0,1\}^L$.*

- *t-**Validity:** If the sender P_s is honest, then all 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'.*

Protocol Π is called a (t_s, t_a)-perfectly-secure network-agnostic reliable broadcast protocol, if in the network agnostic setting, it achieves t_s-validity and t_s-consistency properties when the network is synchronous, and t_a-validity and t_a-consistency properties when the network is asynchronous.

The authors in [PP24] give a construction of a network-agnostic RBC protocol with $\mathcal{O}(Ln^2)$ communication bits for broadcasting an L-bit message. We use $\texttt{Time}_{\texttt{bc}}$ to denote the running time of Π when the dealer is honest and the network is synchronous.

3.5 Network-Agnostic Byzantine Agreement

Let $L \in \mathbb{N}$ and $\mathcal{M} = \{0,1\}^L \cup \{\perp\}$ denote the input and output space. In a Byzantine agreement (BA) protocol, each party P_i takes a message $m_i \in \mathcal{M}$ as input. 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 2 ([ACC22, PP24]). *Let t be the corruption threshold, $\mathcal{M} = \{0,1\}^L \cup \{\perp\}$ be the input and output space, and $\mathcal{P}$ be the party set. Let Π be a protocol executed by parties in $\mathcal{P}$ with up to t corruptions, each party P_i takes a value $m_i \in \mathcal{M}$ as input and terminates after getting the output $m \in \mathcal{M}$.*

- *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 $\mathcal{M}$.*

We say the protocol Π achieves t-secure when it achieves t-validity, t-consistency, and t-termination properties.

Protocol Π is called a t-perfectly-secure synchronous-BA protocol if it achieves t-secure when the network is synchronous.

Protocol Π is called a t-perfectly-secure asynchronous-BA protocol if it achieves t-secure when the network is asynchronous.

Protocol Π is called a (t_s, t_a)-perfectly-secure network-agnostic BA protocol, if in the network agnostic setting, it achieves both t_s-secure when the network is synchronous, and t_a-secure when the network is asynchronous.

In this work, we assume that $t_a < t_s$. Then we note that an asynchronous BA (ABA) protocol with t_s corruption threshold is sufficient to realize the network-agnostic BA protocol defined above. Some previous works [ACC22, PP24] define the notion of "best-of-both-worlds" BA protocol, which is stronger than our above definition of network-agnostic BA. The difference is that they require that when the network is synchronous, all honest parties should terminate the BA protocol within a fixed time bound. Then an ABA protocol is not sufficient to achieve this goal, since the running time of ABA is not deterministic. To summarize, an ABA protocol with t_s corruption threshold is sufficient to realize our definition of network-agnostic BA protocol, but not sufficient to realize the "best-of-both-worlds" BA protocol [ACC22, PP24].

Although an asynchronous BA (ABA) protocol with t_s corruption threshold is sufficient to be used as a network-agnostic BA protocol in our setting, the current best results of ABA in [BCP20] perform a large additive overhead $\mathcal{O}(n^8 \log n)$ bits and $\mathcal{O}(n)$ rounds in expectation. To improve the efficiency, we propose a new construction of network-agnostic BA protocol in appendix B, and this new construction actually realizes the definition of "best-of-both-worlds" BA [ACC22, PP24]. The achieved communication complexity is $\mathcal{O}(Ln^3 + n^4 \log n)$ bits for L-bit agreement. For the round complexity, when the network is synchronous, we guarantee that parties will terminate after $\mathcal{O}(n)$ rounds. When the network is asynchronous, parties require additional $\mathcal{O}(1)$ rounds in expectation to terminate.

4 Secure Channel with Synchronous Identifiable Abort

4.1 Guarantee a Similar Termination Time for Agreement Protocol in the Synchronous Network

In our later construction, we require that all parties terminate the agreement protocol at a similar time when the network is synchronous. In this subsection, we describe how to ensure that all parties terminate an arbitrary network-agnostic agreement protocol Π at approximately the same time when executed over a synchronous network. We construct a modified protocol Π' as follows. Let Δ denote the maximum message delivery delay in the synchronous network. Then, in the synchronous setting, all parties are guaranteed to terminate Π' within a time difference of at most 2Δ.

Protocol Π'

Let Π denote a network-agnostic agreement protocol, which guarantees all honest parties terminate with the same output and is secure against 1/3 fraction of corrupted parties in the synchronous setting and 1/4 fraction of corrupted parties in the asynchronous setting. Let n denote the number of parties and $t_s < n/3$ denote the number of corrupted parties when the network is synchronous. When the network is asynchronous, we require the number of corrupted parties to be less than t_s.

1: All parties participate in Π after they learn their inputs, and we denote the agreement result of Π as m.
2: Upon terminating Π with m, each party sends $(\texttt{Echo}, m)$ to all parties.
3: Upon receiving messages m in the following ways, send $(\texttt{Ready}, m)$ to all parties:
   – The same $(\texttt{Echo}, m)$ from $2t_s + 1$ distinct parties.
   – The same $(\texttt{Ready}, m)$ from $t_s + 1$ distinct parties.

4: Upon receiving $(\texttt{Ready}, m)$ from $2t_s + 1$ distinct parties, all parties terminate with output m.

We briefly explain why all parties will terminate within 2Δ time delay in the synchronous setting. Note that all parties may participate in the protocol at different times, and they are message-driven:

1. When the first honest party terminates, he has received $(\texttt{Ready}, m)$ from $2t_s + 1$ distinct parties, which means that at least $t_s + 1$ honest parties have sent $(\texttt{Ready}, m)$ to all parties.
2. In the following Δ time delay, the $(\texttt{Ready}, m)$ sent from these honest parties will be delivered to all honest parties, and they will send $(\texttt{Ready}, m)$ to all honest parties.
3. In the following Δ time delay, all honest parties are guaranteed to receive $(\texttt{Ready}, m)$ from $2t_s+1$ distinct parties, and they will terminate with the same message m.

Building on Π' and the network-agnostic RBC from [PP24] and BA protocols from appendix B, we obtain modified protocols $\Pi'_{\texttt{rbc}}$ and $\Pi'_{\texttt{ba}}$ that guarantee all parties terminate at approximately the same time. For an L-bit broadcast or agreement, the resulting communication complexity is $\mathcal{O}(Ln^2)$ bits and $\mathcal{O}(Ln^3+n^4\log n)$ bits, respectively.

4.2 Realization of $\Pi_{\texttt{Channel}}$

As we introduced in Section 2.1, we first consider a P2P message from a sender B to a receiver A. Let $\texttt{rid}$ denote the round id, we assume that all honest parties enter the $\texttt{rid}$th round within a time difference of at most Δ in the synchronous network. The construction of $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$ is as follows. It guarantees that when all parties terminate $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$, all parties will either agree that A receives messages from B or eliminate A and B (or only A).

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

For $A \neq B$, all parties proceed the following steps. If $A = B$, all parties directly terminate with $(\texttt{Confirm}, A, B, \texttt{rid})$.

Sending Message

1: After B enters the $\texttt{rid}$-th round, B sends his message m to A.
2: After A enters the $\texttt{rid}$-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 invokes $\Pi'_{\texttt{rbc}}$ to broadcast a confirmation $(\texttt{Confirm}, A, B, \texttt{rid})$. Otherwise, A invokes $\Pi'_{\texttt{rbc}}$ to broadcast a conflict $(\texttt{Conflict}, A, B, \texttt{rid})$.

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 $4\Delta + \texttt{Time}_{\texttt{bc}}$ for the final broadcast message from A.
   - If the received broadcast message is in the form of $(\texttt{Confirm}, A, B, \texttt{rid})$ or $(\texttt{Conflict}, A, B, \texttt{rid})$, 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 $(\texttt{Conflict}, A, B, \texttt{rid})$, all parties output $(\texttt{Eliminate}, A, B, \texttt{rid})$.
   - Otherwise, all parties output $(\texttt{Eliminate}, A, B, \texttt{rid})$.

During the execution of the synchronous protocol, parties may also need to broadcast the messages in some rounds. Although the broadcast channel can be realized by the P2P channel in the perfectly secure setting, we give a more direct realization of $\Pi_{\texttt{BC-Send}}$ to handle the broadcast message. The construction of $\Pi_{\texttt{BC-Send}}$ is similar to $\Pi_{\texttt{P2P-Send}}$, it takes the identity of party B and the round id $\texttt{rid}$ as input, and B will reliably broadcast his messages at the beginning.

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

Sending Message

1: After B enters the $\texttt{rid}$-th round, B invokes $\Pi'_{\texttt{rbc}}$ to broadcast his message m.
2: For each party A, after he enters the $\texttt{rid}$-th round, A waits for $\Delta + \texttt{Time}_{\texttt{bc}}$ time for the message m from B. If A receives the message from B within the time limit, A invokes $\Pi'_{\texttt{rbc}}$ to broadcast a confirmation $(\texttt{Confirm}, A, B, \texttt{rid})$. Otherwise, A invokes $\Pi'_{\texttt{rbc}}$ to broadcast a conflict $(\texttt{Conflict}, A, B, \texttt{rid})$.

Termination

The termination process is the same as $\Pi_{\texttt{P2P-Send}}$, except that each party P_i waits for $3\Delta + 2\texttt{Time}_{\texttt{bc}}$ for the final broadcast message from A.

Putting it all together. We construct the protocol $\Pi_{\texttt{Channel}}$ from $\Pi_{\texttt{P2P-Send}}$ and $\Pi_{\texttt{BC-Send}}$. Each party takes as input a set $\mathcal{P}'$, which contains the identities of the parties that have been eliminated. For every remaining party $P_i \notin \mathcal{P}'$, it inputs a vector of messages and distributes them using $\Pi_{\texttt{P2P-Send}}$ and $\Pi_{\texttt{BC-Send}}$. Once all parties have terminated the executions of $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$ and $\Pi_{\texttt{BC-Send}}(B, \texttt{rid})$ for $A, B \in \mathcal{P} \setminus \mathcal{P}'$, they either agree on new parties to be eliminated, or all alive parties in $\mathcal{P} \setminus \mathcal{P}'$ confirm that they have successfully received each other's messages.

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

All parties take a set $\mathcal{P}'$ as input. Each party $P_i \notin \mathcal{P}'$ takes a vector of messages $(\texttt{rid}, m_0^{(i)}, \{m_j^{(i)}\}_{j \notin \mathcal{P}'})$ as input.

1: Each party $P_i \notin \mathcal{P}'$ takes $m_0^{(i)}$ as the broadcast message, and $m_j^{(i)}$ as the P2P message for each $P_j \notin \mathcal{P}'$. Then P_i invokes $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$ for every $A, B \in \mathcal{P} \setminus \mathcal{P}'$, and invokes $\Pi_{\texttt{BC-Send}}(B, \texttt{rid})$ for every $B \in \mathcal{P} \setminus \mathcal{P}'$.
2: All parties initialize a set $\mathcal{W} = \varnothing$. Upon terminating $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$ and $\Pi_{\texttt{BC-Send}}(B, \texttt{rid})$ for all $A, B \in \mathcal{P} \setminus \mathcal{P}'$
   – If the outputs are all in the form of $(\texttt{Confirm}, A, B, \texttt{rid})$, all parties proceed.
   – Otherwise, for the first instance of $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$ or $\Pi_{\texttt{BC-Send}}(B, \texttt{rid})$ that terminates with $(\texttt{Eliminate}, A, B, \texttt{rid})$, all parties add the identities of (A, B) to $\mathcal{W}$.
3: All parties terminate with $\mathcal{W}$, which either is $\varnothing$ or contains a pair of conflicted parties. If $\mathcal{W} = \varnothing$, then each party $P_i \in \mathcal{P}'$ additionally terminates with $(\texttt{rid}, \mathcal{W}, \{(m_0^{(j)}, m_i^{(j)})\}_{j \notin \mathcal{P}'})$.

Optimization. In the above construction, all parties need to invoke $\Pi'_{\texttt{ba}}$ in each call of $\Pi_{\texttt{P2P-Send}}$ and $\Pi_{\texttt{BC-Send}}$, which adds up to $O(n^2)$ calls of $\Pi'_{\texttt{ba}}$ in each round. We show a simple optimization that reduces the number of calls of $\Pi'_{\texttt{ba}}$ and $\Pi'_{\texttt{rbc}}$ to $O(n)$.

In each round, for each party A, we only invoke a single call of $\Pi_{\texttt{ba}}$. Each party P_i sets his input to be 1 if he received the broadcast message from A in $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$ and $\Pi_{\texttt{BC-Send}}(A, \texttt{rid})$ for all B within the time-bound, and sets his input to be 0 otherwise. Then P_i follows the termination phase of each $\Pi_{\texttt{P2P-Send}}(A, B, \texttt{rid})$ and $\Pi_{\texttt{BC-Send}}(A, \texttt{rid})$ to decide the output. We can similarly show that 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.

Based on this optimization, let $L_{\texttt{bc}}, L_{\texttt{P2P}}$ denote be the total broadcast message size and P2P message size, the communication complexity of $\Pi_{\texttt{Channel}}$ is $\mathcal{O}(L_{\texttt{P2P}} + L_{\texttt{bc}} n^2 + n^5 \log n)$ bits.

For the round complexity, it mainly comes from n parallel instances of $\Pi'_{\texttt{ba}}$. For each of $\Pi'_{\texttt{ba}}$, it costs $\mathcal{O}(n)$ deterministic rounds plus additional $\mathcal{O}(1)$ rounds in expectation (when the network is asynchronous) to terminate. Then running these n instances of $\Pi'_{\texttt{ba}}$ in parallel would cost $\mathcal{O}(n)$ deterministic rounds plus $\mathcal{O}(\log n)$ rounds in expectation (when the network is asynchronous) to terminate.

5 Construction of Perfectly Secure Agnostic MPC

In this section, we present our construction of network-agnostic MPC. We begin with a perfectly secure agnostic triple-generation protocol, which serves as the foundation for realizing the full agnostic MPC protocol.

5.1 Network-Agnostic Triple Generation Compiler

In this subsection, we give the construction of our triple-generation protocol. Upon terminating the protocol, all parties will output a party set $\mathcal{P}'$, and parties in $\mathcal{P}'$ get their shares of random Beaver triples. If $|\mathcal{P}'| > 4t_a + 1$, then parties in $\mathcal{P}'$ get degree-t_s random Beaver triples. Otherwise, $|\mathcal{P}'| = 4t_a + 1$, and parties in $\mathcal{P}'$ get degree-t_a random Beaver triples.

To build this triple-generation protocol, we rely on two sub-protocols $\Pi_{\texttt{Triple-Id-Sync}}$ and $\Pi_{\texttt{Triple-Async}}$, which realize the functionalities $\mathcal{F}_{\mathsf{Triple\text{-}Id\text{-}Sync}}$ and $\mathcal{F}_{\mathsf{Triple\text{-}Async}}$ defined below, respectively. Let N denote the number of triples to be prepared, for the instantiation of these two functionalities:

- For $\mathcal{F}_{\mathsf{Triple\text{-}Id\text{-}Sync}}$, we use the construction in [BH08] to build $\Pi_{\texttt{Triple-Id-Sync}}$, which securely computes $\mathcal{F}_{\mathsf{Triple\text{-}Id\text{-}Sync}}$ against $1/3$ corruptions. The communication complexity is $\mathcal{O}((Nn+n^2)\log n)$ P2P communication bits plus $\mathcal{O}(\log N + n\log n)$ broadcast bits.[4] The round complexity for P2P communication is $\mathcal{O}(1)$.
- For $\mathcal{F}_{\mathsf{Triple\text{-}Async}}$, we can use the construction in [AAPS25] to build $\Pi_{\texttt{Triple-Async}}$, which securely computes $\mathcal{F}_{\mathsf{Triple\text{-}Async}}$ against $1/4$ corruptions. The communication complexity is $\mathcal{O}((Nn+n^4)\log n)$ bits, and the expected round complexity is $\mathcal{O}(1)$.

Functionality $\mathcal{F}_{\mathsf{Triple\text{-}Id\text{-}Sync}}$

$\mathcal{F}_{\mathsf{Triple\text{-}Id\text{-}Sync}}$ runs with a party set $\mathcal{P}'$, and an adversary $\mathcal{S}$. Denote N as the number of random degree-t_s Beaver triples to be prepared.

1: Receive the set $\mathcal{C}orr$ of corrupted parties. Let $t := |\mathcal{C}orr|$ and $n := |\mathcal{P}'|$, if $t_s + 2t < n$, $\mathcal{F}_{\mathsf{Triple\text{-}Id\text{-}Sync}}$ proceeds. Otherwise, $\mathcal{F}_{\mathsf{Triple\text{-}Id\text{-}Sync}}$ gives up security and sends all views to $\mathcal{S}$.
2: For all $i \in [N]$, randomly samples a_i, b_i, c_i such that $c_i = a_i \cdot b_i$.
3: For all $i \in [N]$, receive a set of shares $\{u_{i,j}, v_{i,j}, w_{i,j}\}_{j\in\mathcal{C}orr}$ of corrupted parties from $\mathcal{S}$. Then sample three random degree-t_s Shamir sharings $([a_i]_{t_s}, [b_i]_{t_s}, [c_i]_{t_s})$ based on the shares of corrupted parties and the secrets a_i, b_i, c_i.
4: Send a request-based delayed output of shares of $\{([a_i]_{t_s}, [b_i]_{t_s}, [c_i]_{t_s})\}_{i=1}^{N}$ to all parties in $\mathcal{P}$.
   – Upon receiving a request $(\texttt{Conflict}, P_i, P_j)$ from $\mathcal{S}$, if the output of all parties has not been delivered and $P_i, P_j \in \mathcal{P}$, change it to $(\texttt{Conflict}, P_i, P_j)$. Otherwise, ignore this message.

Functionality $\mathcal{F}_{\mathsf{Triple\text{-}Async}}$

$\mathcal{F}_{\mathsf{Triple\text{-}Async}}$ runs with a party set $\mathcal{P}'$, and an adversary $\mathcal{S}$. Denote N as the number of random degree-t_a Beaver triples to be prepared.

1: Receive the set $\mathcal{C}orr$ of corrupted parties in $\mathcal{P}$. Let $t := |\mathcal{C}orr|$ and $n := |\mathcal{P}'|$, if $t \le t_a$ and $4t_a < n$, $\mathcal{F}_{\mathsf{Triple\text{-}Async}}$ proceeds. Otherwise, $\mathcal{F}_{\mathsf{Triple\text{-}Async}}$ gives up security and sends all views to $\mathcal{S}$.
2: For all $i \in [N]$, randomly samples a_i, b_i, c_i such that $c_i = a_i \cdot b_i$.
3: For all $i \in [N]$, receive a set of shares $\{u_{i,j}, v_{i,j}, w_{i,j}\}_{j\in\mathcal{C}orr}$ of corrupted parties from $\mathcal{S}$. Then sample three random degree-t_a Shamir sharings $([a_i]_{t_a}, [b_i]_{t_a}, [c_i]_{t_a})$ based on the shares of corrupted parties and the secrets a_i, b_i, c_i.
4: Send a request-based delayed output of shares of $\{([a_i]_{t_a}, [b_i]_{t_a}, [c_i]_{t_a})\}_{i=1}^{N}$ to all parties in $\mathcal{P}$.

Then we give our construction of $\Pi_{\texttt{NA-Triple}}$ for triple generation. Let N be the number of triples to be prepared, the communication complexity is $\mathcal{O}((Nn + n^6)\log n)$ bits. The total round complexity is $\mathcal{O}(n^2)$, since there are $\mathcal{O}(n)$ segments, and in each segment parties invoke $\Pi_{\texttt{Channel}}$, which requires $\mathcal{O}(n)$ rounds plus additional $\mathcal{O}(\log n)$ rounds in expectation (when the network is asynchronous).

[4] We use a variant of [BH08], which uses $\mathcal{O}(n)$ instances of broadcast to replace the BA protocol. This helps to reduce the round complexity of P2P communication to a constant, and additional require each party to invoke $\mathcal{O}(1)$ instances of broadcast. Refer to Appendix A for more details.

Protocol $\Pi_{\texttt{NA-Triple}}$

Let $\Pi_{\texttt{Triple-Id-Sync}}$ be a synchronous Beaver triple generation protocol that securely computes $\mathcal{F}_{\texttt{Triple-Id-Sync}}$, $\Pi_{\texttt{Triple-Async}}$ be an asynchronous Beaver triple generation protocol that securely computes $\mathcal{F}_{\texttt{Triple-Async}}$. Suppose R is the (upper bound of the) number of rounds required by $\Pi_{\texttt{Triple-Id-Sync}}$.
All parties define $\mathcal{R} = \{(\Pi_{\texttt{Triple-Id-Sync}}, j)\}_{j\in[R]}$ and initialize a set $\mathcal{W} = \varnothing$. They also take a public parameter N as input, which denotes the number of random Beaver triples to be prepared.

Regular Procedure

In this procedure, all parties aim to generate N degree-t_s random Beaver triples. They first divide the generation of N degree-t_s Beaver triples into n segments. All parties execute these n segments in sequence. In each segment, from $\texttt{rid} = 1$ to R, parties not in $\mathcal{W}$ execute the following steps to generate N/n degree-t_s Beaver triples for all parties in $\mathcal{P} \setminus \mathcal{W}$. All parties initialize the total party number $n' = n$ and corruption threshold $t' = t_s$.

1: Each party $P_i \notin \mathcal{W}$ follows the protocol $\Pi_{\texttt{Triple-Id-Sync}}$ with party number n' and corruption threshold t' to compute his $\texttt{rid}$-th round broadcast message $m_0^{(i)}$ and P2P messages $\{m_j^{(i)}\}_{j\notin\mathcal{W}}$, where each $m_j^{(i)}$ is the P2P message to P_j.

2: All parties invoke an instance of $\Pi_{\texttt{Channel}}$ with input $\mathcal{W}$, each party $P_i \notin \mathcal{W}$ takes input $(\texttt{rid}, m_0^{(i)}, \{m_j^{(i)}\}_{j\notin\mathcal{W}})$. Upon receiving the output $\mathcal{W}'$ from $\Pi_{\texttt{Channel}}$, all parties compute $\mathcal{W} \leftarrow \mathcal{W}\cup\mathcal{W}', n' \leftarrow n' - |\mathcal{W}'|, t' \leftarrow t' - |\mathcal{W}'|/2$.

3: All parties first check whether $|\mathcal{W}| = 2(t_s - t_a)$. If true, they move to the fallback procedure with the set $\mathcal{W}$. Otherwise, depending on the size of $\mathcal{W}', \mathcal{W}$ and $\texttt{rid}$, all parties check whether $\mathcal{W}' = \varnothing$,

- If true, then if $\texttt{rid} < R$, they move to the $(\texttt{rid}+1)$-th round of $\Pi_{\texttt{Triple-Id-Sync}}$. Otherwise, the $\texttt{rid} = R$, all parties compute their output following $\Pi_{\texttt{Triple-Id-Sync}}$ and move to the next segment. For the output of $\Pi_{\texttt{Triple-Id-Sync}}$,
  - If the output is $(\texttt{Conflict}, P_i, P_j)$, they add the identities of P_i, P_j to set $\mathcal{W}$ and execute the current segment again.
  - Otherwise, let $\mathcal{P}' = \mathcal{P} \setminus \mathcal{W}$, all parties in $\mathcal{P}'$ have generated valid degree-t_s Beaver triples. Then all parties record the party set $\mathcal{P}'$, and parties in $\mathcal{P}'$ record their shares of N/n degree-t_s Beaver triples.
- Otherwise, they execute the current segment again.

If all parties eventually succeed in generating N degree-t_s triples in the Regular Procedure, we set $\mathcal{P}'$ to be the party set that all parties agreed on in the last segment, and parties in $\mathcal{P}'$ hold shares of N degree-t_s Beaver triples. All parties terminate with the party set $\mathcal{P}'$, and parties in $\mathcal{P}'$ terminate with their shares of N degree-t_s Beaver triples.

Fallback Procedure

Let $\mathcal{P}'$ be the set of the first $4t_a + 1$ parties in $\mathcal{P} \setminus \mathcal{W}$, all parties in $\mathcal{P}'$ invoke $\Pi_{\texttt{Triple-Async}}$ with party number $4t_a + 1$ and corruption threshold t_a to generate N degree-t_a Beaver triples for parties in $\mathcal{P}'$. Then all parties output the set $\mathcal{P}'$, parties in $\mathcal{P}'$ output their shares of degree-t_a Beaver triples. Note that all parties will execute this step assuming that the network is asynchronous.

Analysis of the Communication Complexity. The total communication costs of $\Pi_{\texttt{NA-Triple}}$ includes the costs of $\Pi_{\texttt{Triple-Id-Sync}}$, $\Pi_{\texttt{Channel}}$, and the instantiation of $\mathcal{F}_{\texttt{Triple-Async}}$. For each instance of $\Pi_{\texttt{Triple-Id-Sync}}$, the P2P communication bit is $\mathcal{O}((N + n^2)\log n)$ and the broadcast communication bit is $\mathcal{O}(\log N + n\log n)$, and there are $\mathcal{O}(1)$ P2P communication rounds. Note that $n^2 \log N \le N \log n^2$ when $N \ge n^2$, then the total cost of $\Pi_{\texttt{Channel}}$ for each instance of $\Pi_{\texttt{Triple-Id-Sync}}$ is $\mathcal{O}((N + n^5)\log n)$ bits. Since the communication cost of instantiating $\mathcal{F}_{\texttt{Triple-Async}}$ is $\mathcal{O}((Nn + n^4)\log n)$ bits and there are $\mathcal{O}(n)$ instances of $\Pi_{\texttt{Triple-Id-Sync}}$, then the total communication complexity is $\mathcal{O}((Nn + n^6)\log n)$ bits.

5.2 Efficient Public Reconstruction

As we introduced in Section 2.3, we give the public reconstruction protocol $\Pi_{\texttt{PubRec}}$ that will be used in the online phase. Let N be the number of sharings to be reconstructed, the communication complexity is $\mathcal{O}((Nn + n^2)\log n)$ bits.

Protocol $\Pi_{\texttt{PubRec}}$

All parties in $\mathcal{P}$ take a party set $\mathcal{P}'$ as input. Let N be the number of shares to be reconstructed, $\alpha_1, \ldots, \alpha_n$ be distinct field elements, t' denote the number of corrupted parties in $\mathcal{P}'$, t denote the degree of Shamir sharings to be reconstructed, and $n' = |\mathcal{P}'|$. The parameter t', t, n' satisfies the relation $(n' - t - 1)/2 \geq t'$. All parties take their shares of $[x_1]_t, \ldots, [x_N]_t$ as inputs.

1: Divide $[x_1]_t, \ldots, [x_N]_t$ into $N/(t+1)$ groups, each of size $t+1$. For each group, each party $P_i \in \mathcal{P}$ does the following.

(1). Let $[s^{(0)}]_t, \ldots, [s^{(t)}]_t$ denote the degree-t Shamir secret sharings in this group. We define a degree-t polynomial $f(X)$ as follows.

$$f(X) = s^{(0)} + s^{(1)} \cdot X + \cdots + s^{(t)} \cdot X^t$$

Then if $P_i \in \mathcal{P}'$, he sends his share of $[f(\alpha_j)]_t = [s^{(0)}]_t + [s^{(1)}]_t \cdot \alpha_j + \cdots + [s^{(t)}]_t \cdot \alpha_j^t$ to each $P_j \in \mathcal{P}'$. Otherwise, P_i does nothing in this step.

(2). Each party $P_i \in \mathcal{P}'$ uses online error correction on shares of $[f(\alpha_i)]_t$ received from parties in $\mathcal{P}'$ to reconstruct $f(\alpha_i)$. Then P_i sends $f(\alpha_i)$ to all parties in $\mathcal{P}$.

(3). Each party $P_i \in \mathcal{P}$ uses online error correction on $f(\alpha_j)$ received from $P_j \in \mathcal{P}'$ to reconstruct $f(X)$. Then P_i records $s^{(0)}, \ldots, s^{(t)}$.

2: All parties in $\mathcal{P}$ output $x_1, \ldots, x_N$.

5.3 Construction of Network-Agnostic MPC

We give the construction of $\Pi_{\texttt{NA-MPC}}$ to realize a perfectly secure agnostic MPC based on $\Pi_{\texttt{NA-Triple}}$. Let $|C|$ denote the circuit size, D denote the circuit depth, and C_I denote the input size. The achieved communication complexity is $\mathcal{O}((|C|n+n^6)\log n)$ bits in the offline phase and $\mathcal{O}((|C|n+(D+C_I)n^2+n^5)\log n)$ in the online phase. When the network is synchronous, the round complexity is $\mathcal{O}(D+n^2)$ plus additional $\mathcal{O}(1)$ rounds in expectation. When the network is asynchronous, it requires additional $\mathcal{O}(n \log n)$ rounds in expectation.

A subtle issue in the input phase is that the parties not only agree on a common set of dealers that have distributed their inputs, but also ensure that, in the synchronous case, all honest parties are included in this set. To address this, we introduce a waiting period so that every honest party can complete input distribution in the synchronous case, and only then run agreement to determine the common set. Refer to the following detailed construction.

Protocol $\Pi_{\texttt{NA-MPC}}$

Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ denote the set of all parties, $|C|$ denote the circuit size, $|C_I|$ denote all parties' input size, and t_s, t_a be the corresponding corruption threshold when the network is synchronous or asynchronous. We use $\Pi_{\texttt{ba}}, \Pi_{\texttt{rbc}}$ to denote arbitrary network-agnostic BA and RBC protocols.

Offline Phase

Preparation of Beaver Triples. All parties start by invoking the regular procedure of $\Pi_{\texttt{NA-Triple}}$ with input parameter $|C| + |C_I|$ to prepare random Beaver triples. All parties will terminate $\Pi_{\texttt{NA-Triple}}$ with a set $\mathcal{P}'$:

- If all parties terminate the regular phase, parties in $\mathcal{P}'$ get $|C| + |C_I|$ degree-t_s random Beaver triples.
- Otherwise, all parties terminate the fallback procedure, parties in $\mathcal{P}'$ get $|C| + |C_I|$ degree-t_a random Beaver triples.

For the first $|C|$ random Beaver triples, we assign each of them to a multiplication gate. For the last $|C_I|$ Beaver triples, parties in $\mathcal{P}'$ only record the first random sharings in each triple. These random sharings will be used when parties distribute their inputs. In the following, we will use t to denote the degree of random Beaver triples that all parties prepared in the offline phase.

Online Phase

1: **Initialization.** All parties first invoke an instance of $\Pi'_{\texttt{ba}}$ with input 1. Upon terminating $\Pi'_{\texttt{ba}}$ with output 1, we know that when the network is synchronous, different honest parties will terminate $\Pi'_{\texttt{ba}}$ at a similar time (at most 2Δ difference), where Δ is the time bound for each round in the synchronous case.

2: **Distributing Inputs.** For each party P_i's input x_i, all parties assign an unused degree-t sharing $[r]_t$ for it. Then parties in $\mathcal{P}'$ send their shares of $[r]_t$ to P_i, and P_i uses online error correction to reconstruct r. Upon getting r, P_i invokes $\Pi_{\texttt{rbc}}$ to broadcast $x_i + r$. Upon receiving $x_i + r$ from P_i, parties in $\mathcal{P}'$ locally compute $[x_i]_t = x_i + r - [r]_t$.

3: **Agreement on the Set of Inputs.** All parties initialize n instances of the network-agnostic $\Pi_{\texttt{ba}}$, one for each party P_i, denoted by $\Pi_{\texttt{ba}}^{(i)}$. Then, each party proceeds to the following steps after waiting for an additional $\texttt{Time}_{\texttt{bc}} + 2\Delta$ once it terminates $\Pi'_{\texttt{ba}}$ in Step 1:

(1) Once a party terminates the P_i's $\Pi_{\texttt{rbc}}$, each party sets its input to $\Pi_{\texttt{ba}}^{(i)}$ as 1.

(2) After a party has terminated at least $n - t_s$ instances of $\Pi_{\texttt{ba}}$ with output 1, for all P_i that this party has not terminated P_i's $\Pi_{\texttt{rbc}}$, it sets its input to $\Pi_{\texttt{ba}}^{(i)}$ as 0.

(3) When all $\Pi_{\texttt{ba}}$ instances have terminated, all parties define $\mathcal{D}$ as the set of identities corresponding to the all instances of $\Pi_{\texttt{ba}}$ that output 1.

For every $P_i \in \mathcal{D}$, all parties hold the shares $[x_i]_t$ of its input. For every $P_i \notin \mathcal{D}$, the parties set the shares of P_i's input to 0.

4: **Online Circuit Evaluation.** All parties in $\mathcal{P}'$ evaluate the circuit gate by gate:

- For every addition gate with input sharings $[x]_t, [y]_t$, all parties in $\mathcal{P}'$ locally compute $[z]_t = [x]_t + [y]_t$.
- For a group of at most $t_s + 1$ multiplication gates, suppose the input degree-t Shamir sharings are denoted by $([x_i]_t, [y_i]_t)_{i=1}^{t+1}$. Let $([a_i]_t, [b_i]_t, [c_i]_t)_{i=1}^{t+1}$ denote the random Beaver triples assigned to these $t + 1$ gates.
  1. All parties in $\mathcal{P}'$ locally compute $[x_i + a_i]_t = [x_i]_t + [a_i]_t$ and $[y_i + b_i]_t = [y_i]_t + [b_i]_t$ for all $i \in [t+1]$.
  2. All parties in $\mathcal{P}'$ invoke $\Pi_{\texttt{PubRec}}$ with their shares and the set $\mathcal{P}'$ to reconstruct $\{x_i + a_i, y_i + b_i\}_{i=1}^{t+1}$.
  3. For all $i \in [t + 1]$, all parties in $\mathcal{P}'$ locally compute:

$$[z_i]_t = (x_i + a_i)(y_i + b_i) - (x_i + a_i)[b_i]_t - (y_i + b_i)[a_i]_t + [c_i]_t.$$

5: **Output and Termination.** All parties do the following steps.

(1). For all output sharings $[\boldsymbol{y}]_t$, all parties in $\mathcal{P}$ invoke $\Pi_{\texttt{PubRec}}$ with input set $\mathcal{P}'$, and parties in $\mathcal{P}'$ additionally take their shares of $[\boldsymbol{y}]_t$ as inputs. All parties in $\mathcal{P}$ will eventually get the output $\boldsymbol{y}$ from $\Pi_{\texttt{PubRec}}$.

Analysis of the Communication Complexity. During the offline phase, parties invoke $\Pi_{\texttt{NA-Triple}}$ to generate triples, which costs $\mathcal{O}(((C+C_I)n+n^6)\log n)$-bit communication. During the input phase, parties need to invoke n instances of $\Pi_{\texttt{ba}}$, n instances of $\Pi_{\texttt{rbc}}$, and one instance of $\Pi'_{\texttt{ba}}$, which costs $\mathcal{O}((C_I n^2 + n^5)\log n)$-bit communication. During the evaluation and output phase, parties only need to do public reconstruction and therefore cost $\mathcal{O}((Cn + Dn^2)\log n)$-bit communication. Then the total communication complexity of $\Pi_{\texttt{NA-MPC}}$ is $\mathcal{O}((Cn + (C_I + D)n^2 + n^6)\log n)$ bits.

Lemma 1. *Protocol $\Pi_{\texttt{NA-MPC}}$ securely computes $\mathcal{F}_{\mathsf{sfe}}$ against a fully malicious adversary $\mathcal{A}$ who corrupts at most t_s parties in the synchronous setting or t_a parties in the asynchronous setting.*

We prove Lemma 1 as follows. The construction of the ideal adversary $\mathcal{S}$ is as follows. 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 $\mathsf{mode} \in \{\mathsf{synch}, \mathsf{asynch}\}$.

Simulator $\mathcal{S}$

Let $\mathcal{C}orr$ denote the set of corrupted parties, $|\mathcal{C}orr| \leq t_s$ when the network is synchronous, and $|\mathcal{C}orr| \leq t_a$ when the network is asynchronous. We denote $\mathcal{S}_{\texttt{TripleIdSync}}, \mathcal{S}_{\texttt{TripleAsync}}$ as the simulator used in the security proofs showing that $\Pi_{\texttt{Triple-Id-Sync}}, \Pi_{\texttt{Triple-Async}}$ realizes $\mathcal{F}_{\texttt{Triple-Id-Sync}}, \mathcal{F}_{\texttt{Triple-Async}}$, respectively. At the beginning of the simulation, $\mathcal{S}$ sends the set of corrupted parties $\mathcal{C}orr$ and the network mode mode to $\mathcal{F}_{\mathsf{sfe}}$.

Offline Phase

1: During the **Regular Procedure**, $\mathcal{S}$ does the following steps. Let $\mathcal{A}_{\texttt{TripleIdSync}}$ be the adversary for the underlying synchronous triple generation protocol $\Pi_{\texttt{Triple-Id-Sync}}$ who just follows the instructions received from the environment $\mathcal{E}$ and passes the messages received from honest parties in each round. $\mathcal{S}$ acts as $\mathcal{E}_{\texttt{TripleIdSync}}$ and $\mathcal{F}_{\texttt{Triple-Id-Sync}}$ and invokes $\mathcal{S}_{\texttt{TripleIdSync}}$.

2: For the simulation of $\mathcal{F}_{\text{Triple-Id-Sync}}$, $\mathcal{S}$ does the following steps.
   (1). Receive the party set $\mathcal{P}'$ and corrupted party set $\mathcal{C}orr'$ from $\mathcal{S}_{\texttt{TripleIdSync}}$.
   (2). Receive a set of shares $\{u_{i,j}, v_{i,j}, w_{i,j}\}_{i\in[N], j\in\mathcal{C}orr'}$ of corrupted parties from $\mathcal{S}_{\texttt{TripleIdSync}}$.
   (3). When all honest parties can get their shares of triples, simulate the behavior of sending output messages from $\mathcal{F}_{\text{Triple-Id-Sync}}$ to all honest parties in $\mathcal{P}'$. Upon receiving a request $(\texttt{Conflict}, P_i, P_j)$ from $\mathcal{S}$, if the output of all honest parties has not been delivered and $P_i, P_j \in \mathcal{P}'$, change it to $(\texttt{Conflict}, P_i, P_j)$. Otherwise, ignore this message.
3: In each segment of triple generation, we set $\mathcal{P}' = \mathcal{P} \setminus \mathcal{W}$, and let $\mathcal{C}orr'$ denote the corrupted parties in $\mathcal{P}'$. Then from $\texttt{rid} = 1$ to R in the current segment,
   (1). $\mathcal{S}$ receives from $\mathcal{A}_{\texttt{TripleIdSync}}$ the P2P messages and broadcast messages that parties in $\mathcal{P}' \setminus \mathcal{C}orr'$ send to parties in $\mathcal{C}orr'$.
   (2). In $\Pi_{\texttt{NA-Triple}}$, for each honest party $P_i \in \mathcal{P}'$, after P_i enters the $\texttt{rid}$th round, $\mathcal{S}$ prepares $(m_0^{(i)}, \{m_j^{(i)}\}_{j\notin\mathcal{W}})$ as follows: $\mathcal{S}$ sets $m_0^{(i)}$ as the broadcast messages of P_i received from $\mathcal{A}_{\texttt{TripleIdSync}}$. 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 $\mathcal{A}_{\texttt{TripleIdSync}}$. Then $\mathcal{S}$ takes $(m_0^{(i)}, \{m_j^{(i)}\}_{j\notin\mathcal{W}})$ as P_i's input for $\Pi_{\texttt{Channel}}$ and honestly execute P_i during the $\Pi_{\texttt{Channel}}$.
   (3). In $\Pi_{\texttt{NA-Triple}}$, after all parties terminate $\Pi_{\texttt{Channel}}$ with a set $\mathcal{W}'$, $\mathcal{S}$ follows the protocol to update $\mathcal{W}$. Then, depending on the size of the sets $\mathcal{W}', \mathcal{W}$ and round id $\texttt{rid}$, if $|\mathcal{W}| = 2(t_s - t_a)$, $\mathcal{S}$ breaks the loop. Otherwise, $\mathcal{S}$ furthers checks whether $\mathcal{W}' = \varnothing$,
      - If true, then if $\texttt{rid} < R$, $\mathcal{S}$ moves to the next round $\texttt{rid} + 1$. Otherwise,
        * If $\mathcal{S}$ has received $(\texttt{Conflict}, P_i, P_j)$ during the simulation of $\mathcal{F}_{\text{Triple-Id-Sync}}$, he follows the protocol to update $\mathcal{W}$ and reruns the current segment.
        * Otherwise, $\mathcal{S}$ considers that all parties can terminate the current segment, and $\mathcal{S}$ learns corrupted parties' shares of triples in this segment. Then $\mathcal{S}$ moves to the next segment.
      - Otherwise, $\mathcal{S}$ follows the protocol to update $\mathcal{W}$ and reruns the current segment.

   If $\mathcal{S}$ terminates the generation of triples for all n segments, $\mathcal{S}$ learns corrupted parties' shares of degree-t_s Beaver triples and moves to the online phase. If $\mathcal{S}$ breaks the loop, $\mathcal{S}$ moves to the fallback procedure.
4: When all parties move to the **Fallback Procedure**, $\mathcal{S}$ does the following steps. $\mathcal{S}$ sets $\mathcal{P}'$ as the first $4t_a + 1$ parties in $\mathcal{P} \setminus \mathcal{W}$. To simulate the fallback procedure, let $\mathcal{A}_{\texttt{TripleAsync}}$ be the adversary for the underlying asynchronous triple generation protocol $\Pi_{\texttt{Triple-Async}}$ who follows the instructions received from the environment and passes the messages received from honest parties in each round. $\mathcal{S}$ acts as $\mathcal{E}_{\text{asynch}}$ and $\mathcal{F}_{\text{Triple-Async}}$ and invokes $\mathcal{S}_{\texttt{TripleAsync}}$.
5: For the simulation of $\mathcal{F}_{\text{Triple-Async}}$,
   (1). Define $\mathcal{C}orr'$ be the set of corrupted parties in $\mathcal{P}'$.
   (2). Receive a set of shares $\{u_{i,j}, v_{i,j}, w_{i,j}\}_{i\in[N], j\in\mathcal{C}orr'}$ of corrupted parties from $\mathcal{S}_{\texttt{TripleAsync}}$.
   (3). When all honest parties can get their shares of triples, simulate the behavior of sending output messages from $\mathcal{F}_{\text{Triple-Async}}$ to all honest parties in $\mathcal{P}'$.
6: For $\mathcal{E}_{\text{asynch}}$, $\mathcal{S}$ instructs $\mathcal{A}_{\texttt{TripleAsync}}$ to corrupt parties in $\mathcal{C}orr'$. Then,
   - Whenever $\mathcal{E}$ activates a party $P_i \in \mathcal{P}'$ in $\Pi_{\texttt{NA-Triple}}$, $\mathcal{S}$ activates P_i in $\Pi_{\texttt{Triple-Async}}$.
   - For each message sent from $P_i \in \mathcal{C}orr'$ to $P_j \in \mathcal{P}' \setminus \mathcal{C}orr'$ in $\Pi_{\texttt{NA-Triple}}$, $\mathcal{S}$ receives the message from $\mathcal{A}$ and instructs $\mathcal{A}_{\texttt{TripleAsync}}$ to let P_i sends the same message to P_j.
   - For each message sent from $P_i \in \mathcal{P}' \setminus \mathcal{C}orr'$ to $P_j \in \mathcal{C}orr'$ in $\Pi_{\texttt{Triple-Async}}$, $\mathcal{S}$ receives the message from $\mathcal{A}_{\texttt{TripleAsync}}$ and sends it to P_j on behalf of P_i in $\Pi_{\texttt{Triple-Async}}$.

   Upon terminating $\Pi_{\texttt{Triple-Async}}$, $\mathcal{S}$ learns corrupted parties' shares of degree-t_a Beaver triples and moves to the online phase.

Online Phase

1: Upon terminating the offline phase, $\mathcal{S}$ learns the set $\mathcal{P}'$. We use $\mathcal{C}orr'$ to denote the corrupted parties set in $\mathcal{P}'$ and set $|\mathcal{C}orr'| = t' \le t$, where t is the degree of the random Beaver triples all parties prepared during the offline phase. Let $\mathcal{H}_{\mathcal{C}}$ denote the set of first $t - t'$ honest parties in $\mathcal{P}'$.
2: During the **Initialization**, $\mathcal{S}$ honestly execute honest parties to participate in Π'_{ba}.
3: During the **Distributing Inputs**, for each party $P_i \in \mathcal{P}$:
   - If P_i is corrupted, $\mathcal{S}$ randomly samples the whole sharing $[r]_t$ based on shares of corrupted parties. Then $\mathcal{S}$ honestly executes each honest party to reconstruct r to $\mathcal{S}$. Upon receiving $x_i + r$ from P_i's reliable broadcast, $\mathcal{S}$ extracts P_i's inputs x_i.

– If P_i is honest, $\mathcal{S}$ simulates the behavior of sending messages between honest parties and P_i. For each share received from an honest party, $\mathcal{S}$ considers this share to be correct. For each share received from a corrupted party, $\mathcal{S}$ checks whether it is the same as the one known to $\mathcal{S}$. If true, $\mathcal{S}$ also considers it correct. When P_i receives $2t+1$ correct shares, $\mathcal{S}$ considers P_i succeeds in using online error correction to reconstruct the secret r.

Then, for each honest party $P_i \in \mathcal{P}$, $\mathcal{S}$ randomly samples a value as $x_i + r$ and broadcasts it on behalf of P_i.

4: During the **Agreement on the Set of Inputs**, $\mathcal{S}$ honestly follows the protocol to execute all instances of Π_{ba} with their inputs. Upon terminating, $\mathcal{S}$ learns the set $\mathcal{D}$ of size at least $n - t_s$. Then, $\mathcal{S}$ samples random values for each party in $\mathcal{H}_{\mathcal{C}}$ as their shares of $[x_i]_t$ for each $P_i \in \mathcal{D}$. Finally, $\mathcal{S}$ sends the set $\mathcal{D}$ and x_i for all corrupted $P_i \in \mathcal{D}$ to $\mathcal{F}_{\mathsf{sfe}}$.

5: During the **Online Circuit Evaluation**, for each addition gate, $\mathcal{S}$ follows the protocol to compute the output share for all parties in $\mathcal{C}orr' \cup \mathcal{H}_{\mathcal{C}}$. For multiplication gates:

(1). $\mathcal{S}$ follows the protocol to compute shares of $\{[x_i + a_i]_t, [y_i + b_i]_t\}_{i=1}^{t+1}$ for all parties in $\mathcal{C}orr' \cup \mathcal{H}_{\mathcal{C}}$.

(2). $\mathcal{S}$ randomly samples the whole sharings $\{[x_i + a_i]_t, [y_i + b_i]_t\}_{i=1}^{t+1}$ based on the shares he computed for parties in $\mathcal{C}orr' \cup \mathcal{H}_{\mathcal{C}}$. Then $\mathcal{S}$ gets each shares of $\{[x_i + a_i]_t, [y_i + b_i]_t\}_{i=1}^{t+1}$ for honest parties in $\mathcal{P}' \setminus \mathcal{H}_{\mathcal{C}}$, and can honestly execute them during the $\Pi_{\mathtt{PubRec}}$.

(3). When all parties terminate $\Pi_{\mathtt{PubRec}}$, $\mathcal{S}$ computes the output shares for all parties in $\mathcal{C}orr' \cup \mathcal{H}_{\mathcal{C}}$.

6: During the **Output and Termination**, $\mathcal{S}$ sends output request to $\mathcal{F}_{\mathsf{sfe}}$. Upon receiving $\boldsymbol{y}$ from $\mathcal{F}_{\mathsf{sfe}}$, $\mathcal{S}$ computes the whole sharing $[\boldsymbol{y}]_t$ based on the secrets $\boldsymbol{y}$ and shares of $[\boldsymbol{y}]_t$ for parties in $\mathcal{C}orr' \cup \mathcal{H}_{\mathcal{C}}$. With the whole sharing $[\boldsymbol{y}]_t$, $\mathcal{S}$ honestly executes the rest of the steps. When honest parties terminate the $\Pi_{\mathtt{PubRec}}$ in the output phase, $\mathcal{S}$ delivers the output from $\mathcal{F}_{\mathsf{sfe}}$ to this honest party.

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 the following, we focus on the simulation of the offline phase. Let $\mathcal{A}_{\mathtt{TripleIdSync}}$ be the adversary for the underlying synchronous triple generation protocol $\Pi_{\mathtt{Triple\text{-}Id\text{-}Sync}}$ who just follows the instructions received from the environment $\mathcal{E}$ and passes the messages received from honest parties in each round. $\mathcal{S}$ acts as $\mathcal{E}_{\mathtt{TripleIdSync}}$ and $\mathcal{F}_{\mathsf{Triple\text{-}Id\text{-}Sync}}$ and invokes $\mathcal{S}_{\mathtt{TripleIdSync}}$. Let $\mathcal{R} = \{(\Pi_{\mathtt{Triple\text{-}Id\text{-}Sync}}, j)\}_{j\in[R]}$.

$\mathbf{Hyb}_{1.1}$: In this hybrid, we focus on the **Regular Procedure**. For any $\mathtt{rid} \in \mathcal{R}$ and the first segment of triple generation, if honest parties could enter the $\mathtt{rid}$th round of $\Pi_{\mathtt{Triple\text{-}Id\text{-}Sync}}$, $\mathcal{S}$ delays the generation of honest parties' shares of triples after the $\mathtt{rid}$th round of $\Pi_{\mathtt{Triple\text{-}Id\text{-}Sync}}$. $\mathcal{S}$ also does not use honest parties' inputs to generate their P2P and broadcast messages in the $\mathtt{rid}$th round, but receives them from $\mathcal{A}_{\mathtt{TripleIdSync}}$ and uses them as honest parties' inputs for $\Pi_{\mathtt{Channel}}$ in this round.

We first show that there exists $\mathcal{S}_{\mathtt{TripleIdSync}}$ and therefore $\mathcal{S}$ can receive the messages from $\mathcal{A}_{\mathtt{TripleIdSync}}$. Assuming that all parties have eliminated $2k$ parties, where $k \in [0, t_s - t_a]$, then there are remaining $n' = n - 2k$ parties. Since for each time when parties eliminate a pair of parties in $\mathcal{W}'$, there is at least one corrupted party when the network is synchronous. Then we guarantee that there are at most $t' = t_s - k$ corrupted parties in $\mathcal{P} \setminus \mathcal{W}$ when the network is synchronous. Then, since the corruption threshold holds $t' < n'/3$, $\Pi_{\mathtt{Triple\text{-}Id\text{-}Sync}}$ can realize $\mathcal{F}_{\mathsf{Triple\text{-}Id\text{-}Sync}}$ with perfect security in the synchronous setting, and there exists $\mathcal{S}_{\mathtt{TripleIdSync}}$.

Then we show that using the honest parties' P2P and broadcast messages received from $\mathcal{A}_{\mathtt{TripleIdSync}}$ does not affect the output distribution. Since $\mathcal{A}_{\mathtt{TripleIdSync}}$ will receive honest P_i's P2P and broadcast messages generated by $\mathcal{S}_{\mathtt{TripleIdSync}}$, and $\Pi_{\mathtt{Triple\text{-}Id\text{-}Sync}}$ realizes $\mathcal{F}_{\mathsf{Triple\text{-}Id\text{-}Sync}}$ with perfect security, then the honest party P_i's P2P and broadcast messages $\mathcal{S}$ received from $\mathcal{A}_{\mathtt{TripleIdSync}}$ have the same distribution as in $\mathbf{Hyb}_{1.\mathtt{rid}-1}$.

Therefore, the distributions of $\mathbf{Hyb}_{1.1}$ and $\mathbf{Hyb}_1$ are the same.

$\mathbf{Hyb}_{1.2}$: In this hybrid, we focus on the **Regular Procedure**. For the first segment of triple generation, if honest parties terminate $\Pi_{\mathtt{Triple\text{-}Id\text{-}Sync}}$ with their shares of triples, $\mathcal{S}$ honestly emulates $\mathcal{F}_{\mathsf{Triple\text{-}Id\text{-}Sync}}$ to generate their shares but delays the generation of their shares until the online phase. Since these triples are not used in the offline phase, this adjustment does not affect the output distribution. The output distributions of $\mathbf{Hyb}_{1.2}$ and $\mathbf{Hyb}_{1.1}$ are the same.

$\mathbf{Hyb}_{1.3}$: In this hybrid, we focus on the **Regular Procedure**. For the rest of the segments of triple generation, $\mathcal{S}$ defers the generation of honest parties' shares of triples until the online phase. During the simulation of the $\texttt{rid}$th round of $\Pi_{\texttt{Triple-Id-Sync}}$ for some $\texttt{rid} \in \mathcal{R}$, $\mathcal{S}$ first receives each honest party's P2P and broadcast messages from $\mathcal{A}_{\texttt{TripleIdSync}}$, and uses them as the input for $\Pi_{\texttt{Channel}}$ of this round to do simulation. Following the same argument in $\mathbf{Hyb}_{1.1}$ and $\mathbf{Hyb}_{1.2}$, the output distributions of $\mathbf{Hyb}_{1.3}$ and $\mathbf{Hyb}_{1.2}$ are the same.

$\mathbf{Hyb}_{1.4}$: In this hybrid, we focus on the **Regular Procedure**. For the instance of $\Pi_{\texttt{Triple-Id-Sync}}$ that honest parties terminate with a set of parties to be eliminated, $\mathcal{S}$ no longer generates honest parties' shares of triples. Since these triples are never used in the online phase, this adjustment does not affect the output distribution. The output distributions of $\mathbf{Hyb}_{1.4}$ and $\mathbf{Hyb}_{1.3}$ are the same.

$\mathbf{Hyb}_{1.5}$: In this hybrid, during the **Fallback Procedure**, we change the construction of $\mathcal{S}$ as above, and let $\mathcal{S}$ play the role of $\mathcal{F}_{\mathsf{Triple\text{-}Async}}$. We first show that in each simulation of $\mathcal{F}_{\mathsf{Triple\text{-}Async}}$, there exists $\mathcal{S}_{\texttt{TripleAsync}}$. Since there are $4t_a + 1$ parties that invoke the protocol $\Pi_{\texttt{Triple-Async}}$ with at most t_a corrupted parties, then these $4t_a + 1$ parties can securely invoke $\Pi_{\texttt{Triple-Async}}$ in a black-box way. Since $\Pi_{\texttt{Triple-Async}}$ realizes $\mathcal{F}_{\mathsf{Triple\text{-}Async}}$ with perfect security, then there exists $\mathcal{S}_{\texttt{TripleAsync}}$ and $\mathcal{S}$ can invoke it. The distributions of $\mathbf{Hyb}_{1.5}$ and $\mathbf{Hyb}_{1.4}$ are the same.

$\mathbf{Hyb}_{1.6}$: In this hybrid, during the **Fallback Procedure**, we defer the generation of honest parties' shares of triples until the online phase. Since these triples are not used in the offline phase, this adjustment does not affect the output distribution. The output distributions of $\mathbf{Hyb}_{1.6}$ and $\mathbf{Hyb}_{1.5}$ are the same.

$\mathbf{Hyb}_{2}$: In the following, we focus on the simulation of the online phase.

$\mathbf{Hyb}_{2.1}$: In this hybrid, we change the generation of random degree-t sharing $[r]_t$ assigned to an honest party in the input phase. $\mathcal{S}$ first randomly samples a value as $x_i + r$, then computes $r = x_i + r - x_i$ and the whole sharing $[r]_t$ based on the secret r and shares of corrupted parties. Since both r and $x_i + r$ are uniformly random, the distributions of $\mathbf{Hyb}_{2.1}$ and $\mathbf{Hyb}_{2}$ are the same.

$\mathbf{Hyb}_{2.2}$: In this hybrid, let $\mathcal{H}_{\mathcal{C}}$ be the set of first $t - t'$ honest parties in $\mathcal{P}'$, $\mathcal{S}$ samples random values as their shares of $[x_i]_t$ for each dealer $P_i \in \mathcal{D}$. Since the whole sharing $[x_i]_t$ is still uniformly random given shares of parties in $\mathcal{C}orr \cup \mathcal{H}_{\mathcal{C}}$, the distributions of $\mathbf{Hyb}_{2.2}$ and $\mathbf{Hyb}_{2.1}$ are the same.

$\mathbf{Hyb}_{2.3}$: In this hybrid, during the **Online Circuit Evaluation**, when all parties need to reconstruct $\{[x_i + a_i]_t, [y_i + b_i]_t\}_{i=1}^{t+1}$, $\mathcal{S}$ first randomly samples the whole sharings $[x_i + a_i]_t, [y_i + b_i]_t$ based on shares of parties in $\mathcal{C}orr \cup \mathcal{H}_{\mathcal{C}}$, then compute $[a_i]_t = [x_i + a_i]_t - [x_i]_t, [b_i]_t = [y_i + b_i]_t - [y_i]_t$. The difference is that we change the generation of $[a_i]_t, [b_i]_t$, however, both of the secrets a_i, b_i and $x_i + a_i, y_i + b_i$ are sampled by $\mathcal{S}$, and they are uniformly random given shares of parties in $\mathcal{C}orr \cup \mathcal{H}_{\mathcal{C}}$. Therefore, the distributions of $\mathbf{Hyb}_{2.3}$ and $\mathbf{Hyb}_{2.2}$ are the same.

$\mathbf{Hyb}_{2.4}$: In this hybrid, while reconstructing the output sharing $[\boldsymbol{y}]_t$ to a corrupted receiver $\mathcal{S}$ first computes $\boldsymbol{y}$ from all parties' inputs and then computes $[\boldsymbol{y}]_t$ from secret $\boldsymbol{y}$ and the shares for parties in $\mathcal{C}orr \cup \mathcal{H}_{\mathcal{C}}$. Finally, $\mathcal{S}$ honestly executes the rest of the steps on behalf of each honest party in $\mathcal{P}'$.

Here, we only change the way of generating the output shares for honest parties not in $\mathcal{H}_{\mathcal{C}}$. We don't change the generation process of shares for parties in $\mathcal{C}orr \cup \mathcal{H}_{\mathcal{C}}$ of $[\boldsymbol{y}]_t$ and the secret $\boldsymbol{y}$, and shares of $[\boldsymbol{y}]_t$ for honest parties not in $\mathcal{H}_{\mathcal{C}}$ are fully determined by the shares for parties in $\mathcal{C}orr \cup \mathcal{H}_{\mathcal{C}}$ and the secret. Therefore, the honest parties' shares of $[\boldsymbol{y}]_t$ stay unchanged. Thus, $\mathbf{Hyb}_{2.4}$ and $\mathbf{Hyb}_{2.3}$ have the same distribution.

$\mathbf{Hyb}_{2.5}$: In this hybrid, $\mathcal{S}$ doesn't generate shares for honest parties out of $\mathcal{H}_{\mathcal{C}}$ of any Shamir secret sharing. For the computation of output sharings reconstructed to corrupted parties, $\mathcal{S}$ doesn't follow the protocol to compute the secrets. Instead, $\mathcal{S}$ directly uses the corrupted parties' outputs as the secrets. Clearly, by the correctness of the protocol, the outputs computed by following the protocol match the outputs from $\mathcal{F}_{\mathsf{sfe}}$. Besides, the shares for honest parties out of $\mathcal{H}_{\mathcal{C}}$ are not used in the simulation. Thus, $\mathbf{Hyb}_{2.5}$ and $\mathbf{Hyb}_{2.4}$ have the same distribution.

$\mathbf{Hyb}_{2.6}$: In this hybrid, $\mathcal{S}$ no longer requires honest parties' inputs. $\mathcal{S}$ first sends the network mode mode, the set of corrupted parties $\mathcal{C}orr$, the set $\mathcal{D}$ and x_i for all corrupted $P_i \in \mathcal{D}$ to $\mathcal{F}_{\mathsf{sfe}}$. When each honest party succeeds in using online error correction to reconstruct $\boldsymbol{y}$ during the $\Pi_{\texttt{PubRec}}$ in the output phase, $\mathcal{S}$ delivers the output from $\mathcal{F}_{\mathsf{sfe}}$ to this honest party. $\mathbf{Hyb}_{2.6}$ and $\mathbf{Hyb}_{2.5}$ have the same distribution.

Note that $\mathbf{Hyb}_{2.6}$ is the ideal-world scenario, then we prove that $\Pi_{\texttt{NA-MPC}}$ realizes $\mathcal{F}_{\mathsf{sfe}}$ with perfect security.

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. C. Liu-Zhang was supported in part by the SNSF grant number 2000-1-243069.

References

[AAPP23] Ittai Abraham, Gilad Asharov, Shravani Patil, and Arpita Patra. Detect, pack and batch: perfectly-secure mpc with linear communication and constant expected time. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 251–281. Springer, 2023.

[AAPP24] Ittai Abraham, Gilad Asharov, Shravani Patil, and Arpita Patra. Perfect asynchronous mpc with linear communication overhead. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 280–309. Springer, 2024.

[AAPS25] Ittai Abraham, Gilad Ashsarov, Arpita Patra, and Gilad Stern. Asynchronous agreement on a core set in constant expected time and more efficient asynchronous vss and mpc. In *Theory of Cryptography Conference*, pages 451–482. Springer, 2025.

[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 *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 92–102, 2022.

[BCG93] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *25th ACM STOC*, pages 52–61. ACM Press, May 1993.

[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.

[BCP20] Laasya Bangalore, Ashish Choudhury, and Arpita Patra. The power of shunning: efficient asynchronous byzantine agreement revisited. *Journal of the ACM (JACM)*, 67(3):1–59, 2020.

[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.

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[BGP92] Piotr Berman, Juan A Garay, and Kenneth J Perry. Bit optimal distributed consensus. In *Computer science: research and applications*, pages 313–321. Springer, 1992.

[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.

[BH08] Zuzana Beerliova-Trubiniova and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In Ran Canetti, editor, *Theory of Cryptography Conference — TCC 2008*, volume 4948 of *Lecture Notes in Computer Science*, pages 213–230. Springer-Verlag, 3 2008.

[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 Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier, editors, *ACM CCS 2025*, pages 1799–1813. ACM Press, October 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.

[BKLZL20] Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 353–380. Springer, Cham, November 2020.

[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.

[Bra84] Gabriel Bracha. An asynchronous [(n - 1)/3]-resilient consensus protocol. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, PODC '84, page 154–162, New York, NY, USA, 1984. Association for Computing Machinery.

[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.

[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.

[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.

[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.

[DDL+24] Sourav Das, Sisi Duan, Shengqi Liu, Atsuki Momose, Ling Ren, and Victor Shoup. Asynchronous consensus without trusted setup or public-key cryptography. *Cryptology ePrint Archive*, 2024.

[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.

[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.

[DN07] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590. Springer, 2007.

[FM88] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 148–161, 1988.

[GLS19] Vipul Goyal, Yanyi Liu, and Yifan Song. Communication-efficient unconditional mpc with guaranteed output delivery. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 85–114, Cham, 2019. Springer International Publishing.

[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.

[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.

[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.

[HMP00] Martin Hirt, Ueli Maurer, and Bartosz Przydatek. Efficient secure multi-party computation. In *International conference on the theory and application of cryptology and information security*, pages 143–161. Springer, 2000.

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

[JLZPS26] Xiaoyu Ji, Chen-Da Liu-Zhang, Daniel Pöllmann, and Yifan Song. Information-theoretic network-agnostic mpc with polynomial communication. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2026.

[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.

[PP24] Shravani Patil and Arpita Patra. Perfectly-secure network-agnostic MPC with optimal resiliency. Cryptology ePrint Archive, Paper 2024/990, 2024.

[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.

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd FOCS*, pages 80–91. IEEE Computer Society Press, November 1982.

A Review of the Triple Generation in [BH08]

In this section, we review the triple generation protocol in [BH08] that achieves security with identifiable abort. We use d to denote the degree of triples to be generated, n to denote the number of parties, and t to denote the corruption threshold among these n parties. The parameters d, n, t satisfy that $t \leq d$ and $d + 2t < n$.

At a high level, there are three main stages for their construction.

- **Stage 1: Generating Non-robust Triples.**
- **Stage 2: Agreement on the Success of Triple Generation.**
- **Stage 3: Fault Localization.**

Realization of Stage 1. Parties first prepare random double sharings in the following form.

$$([a]_d, [a]_t), ([b]_d, [b]_t), ([r]_d, [r]_{2t})$$

The authors in [BH08] show how this can be realized with the help of the hyper-invertible matrix, and we refer to the details of this part to their original paper [BH08]. During the generation of these sharings, parties need to verify whether the degree of each sharing is correct and the secrets in each pair of double sharing are the same. If not, at least one honest party would be "unhappy", indicating the generation of triples fails for this party.

Then all parties use these double sharings to generate a triple in the form of $([a]_d, [b]_d, [c]_d)$, where the first two sharings come from their prepared double sharings, and the goal is to compute the third sharing $[c]_d$ such that $c = a \cdot b$. To achieve this goal, all parties do the following steps.

1. Parties locally compute their shares of $[c]_{2t} = [a]_t \cdot [b]_t$. Then they compute $[e]_{2t} = [c]_{2t} - [r]_{2t}$. Note that if a party is "unhappy" with their shares of the double sharings, it will set its share of $[e]_{2t}$ to be $\perp$.
2. Parties exchange their shares of $[e]_{2t}$ to reconstruct the secret e. During this procedure, if a party receives $\perp$ from any party, it will also be "unhappy". If the received n shares do not lie on a unique degree-$2t$ polynomial, parties also become "unhappy".
3. Upon getting e, parties compute their shares of $[c]_d := [r]_d + e$.

As a result, upon terminating the triple generation protocol, we guarantee that parties can either get their shares of $([a]_d, [b]_d, [c]_d)$ or be "unhappy". Then the next step is to let all parties identify whether all of them are "happy". If true, this means all parties have got their shares of triples, and they can terminate. Otherwise, they need to detect a pair of parties that are responsible for the failure of the triple generation process.

In [BH08], the reconstruction of $[e]_{2t}$ is optimized via batch reconstruction that achieves linear communication.

Realization of Stage 2. In this stage, each party P_i that has got its shares of triples will broadcast $(P_i, \texttt{happy})$, and $(P_i, \texttt{unhappy})$ otherwise. Then parties wait to receive all these broadcast messages. If all of them are in the form of $(P_i, \texttt{happy})$, then they can terminate with their shares of triples. Otherwise, this means at least one party P_i considers the triple generation fails, and parties need to do fault localization and detect the reason why P_i is "unhappy".

We note that in the original construction in [BH08], the authors let all parties first exchange their happy bit with each other and then invoke an instance of the BA protocol to reach an agreement. This only costs $\mathcal{O}(n^2 \log n)$-bit communication. However, the BA protocol would require $\mathcal{O}(n)$ P2P communication rounds to terminate. Here we use n instances of broadcast to replace the BA protocol. Although the communication complexity would blow up to $\mathcal{O}(n^3 \log n)$ bits, the round complexity of P2P communication is reduced to a constant, and each party will invoke one instance of broadcast.

Realization of Stage 3. In this stage, parties proceed with the following steps.

1. Parties first select a party P_r with the smallest index r as the referee, then they send all their views to P_r.
2. With all parties' views, P_r can locally finish all computation for the triple generation procedure. As a result, P_r can detect that there exists a pair of parties (P_i, P_j), and a message with index ℓ where P_i should have sent x to P_j, but P_j claims to have received $x' \neq x$. Then P_r broadcast (ℓ, P_i, P_j, x, x').
3. Upon receiving (ℓ, P_i, P_j, x, x') from P_r, the accused parties P_i and P_j need to broadcast whether they agree with P_r. Let E denote the pair of parties to be eliminated. If P_i disagrees, set $E = \{P_r, P_i\}$. If P_j disagrees, set $E = \{P_r, P_j\}$. Otherwise, set $E = \{P_i, P_j\}$.

For the eliminated parties in E, the authors in [BH08] show that at least one of them is corrupted. Refer to the original paper for more details. During this procedure, each party needs to broadcast $\mathcal{O}(\log n)$ bits, and P_r needs to broadcast $\mathcal{O}(|\ell| + \log n)$ bits.

To summarize, we use a variant of [BH08], and parties only cost $\mathcal{O}(1)$ rounds in P2P communication, and each party will invoke $\mathcal{O}(1)$ instances of broadcast. The total P2P communication complexity is $\mathcal{O}(Nn \log n + n^2 \log n)$ bits, and the broadcast size for P_r is $\mathcal{O}(|\ell| + \log n)$ bits, and the broadcast size for $P_i \neq P_r$ is $\mathcal{O}(\log n)$ bits.

B Instantiation of Network Agnostic Byzantine Agreement

In this section, we show how to instantiate a perfect-secure network-agnostic BA protocol $\Pi_{\texttt{NA-BA}}$ with optimal resilience. The achieved communication complexity is $\mathcal{O}(Ln^3 + n^4 \log n)$ bits for L-bit agreement in expectation. For round complexity, it costs $\mathcal{O}(n)$ deterministic rounds when the network is synchronous. When the network is asynchronous, it requires additional $\mathcal{O}(1)$ rounds in expectation to terminate. Formally, we give the following theorem.

Theorem 3. *Let n, t_s, t_a be positive integers such that $n = 2\max(t_s, t_a) + \max(2t_a, t_s) + 1$. There exists a perfectly secure network-agnostic Byzantine agreement protocol that costs $\mathcal{O}(Ln^3 + n^4 \log n)$ bits of communication in expectation for L-bit agreement. When the network is synchronous, it requires $\mathcal{O}(n)$ rounds to terminate. When the network is asynchronous, it requires additional $\mathcal{O}(1)$ rounds in expectation to terminate.*

Motivation. As we mentioned in section 3, an asynchronous BA protocol with $t_s < n/3$ corruption threshold is sufficient to be used as a network-agnostic BA protocol with optimal corruption threshold in this work. However, the current best result in [BCP20] still costs $\mathcal{O}(Ln^5 \log n + n^8 \log n)$ bits of communication for L-bit agreement in expectation. To make it more efficient, we propose a new construction.

Our Contribution. In this work, we present a new compiler for a perfectly secure network-agnostic BA protocol. Our construction also realizes the notion of "best-of-both-worlds" BA protocol defined in [ACC22, PP24], which is stronger than the network-agnostic BA protocol we defined in 3.[5]

The authors in [ACC22, PP24] also provide compilers for constructing "best-of-both-worlds" BA protocols. However, their constructions rely on an ABA protocol with corruption threshold $t_s < n/3$, whose communication complexity constitutes the main bottleneck. As a comparison, our key departure from the compiler of [ACC22, PP24] is that we employ, in a black-box manner, an ABA protocol tolerating up to $t_a < n/4$ corruption threshold, rather than an ABA protocol tolerating up to $t_s < n/3$ corruption threshold. This choice enables us to leverage the more efficient ABA construction of [AAPS25], which achieves an expected communication complexity of $\mathcal{O}(Ln^2 \log n + n^4 \log n)$ bits for agreeing on an L-bit value.

Note that simply replacing the $t_s < n/3$ ABA used in [PP24] with a $t_a < n/4$ ABA would compromise security. To safely exploit the lower-threshold ABA, we introduce additional mechanisms that ensure its secure integration into the compiler. As a result, our construction inherits the efficiency benefits of the underlying ABA protocol while maintaining perfect security, yielding a more communication-efficient network-agnostic BA protocol.

In the following, we first introduce some preliminaries we used in our construction, then provide an overview of our construction and the details of the construction.

B.1 Preliminaries

In this subsection, we mainly recap the definition of reliable agreement in [Bra84, DDL+24, BKLZL20] and graded consensus in [BCLZL23]. Our construction also relies on the synchronous and asynchronous Byzantine agreement protocols, and we have introduced the property-based definition of them protocol in section 3.

For the synchronous BA protocol used in our construction later, we require it to have an additional property when the network is asynchronous, which we denote weak-validity [DHLZ21].

Definition 3 ([DHLZ21]). *Let t denote the corruption threshold, $\mathcal{M} = \{0,1\}^L \cup \{\perp\}$ be the input and output space, and $\mathcal{P}$ be the party set. Let Π be a protocol executed by parties in $\mathcal{P}$ with up to t corruptions, each party P_i takes a value $m_i \in \mathcal{M}$ as input and terminates after getting the output $m \in \mathcal{M}$.*

- *t-**Validity:** The same as in section 3.5.*
- *t-**Consistency:** The same as in section 3.5.*

[5] Recall that the difference is that, when the network is synchronous, the "best-of-both-worlds" BA protocol requires honest parties to terminate within a fixed time bound.

– t-***Termination:*** *The same as in section 3.5.*
– t-***Weak Validity:*** *If all honest parties hold the same input $m \in \mathcal{M}$ and terminate with y, then $y \in \{\perp, m\}$.*

Protocol Π is called a t_s-perfectly-secure synchronous-BA protocol with t_a asynchronous weak validity if it achieves t_s-validity, t_s-consistency, and t_s-termination properties when the network is synchronous, and t_a-weak validity when the network is asynchronous.

Let $L \in \mathbb{N}$ and $\mathcal{M} = \{0,1\}^L \cup \{\perp\}$ denote the input and output space. In a Reliable Agreement (RA) protocol, each party P_i takes $m_i \in \mathcal{M}$ as input, and all parties output the same string $m \in \mathcal{M}$. We recall the property-based definition of an RA protocol as follows.

Definition 4 ([Bra84, DDL$^+$24]). *Let t be the corruption threshold, $\mathcal{M} = \{0,1\}^L \cup \{\perp\}$ be the input and output space, and $\mathcal{P}$ be the party set. Let Π be a protocol executed by parties in $\mathcal{P}$ with up to t corruptions, each party P_i takes a value $m_i \in \mathcal{M}$ as input and terminates after getting the output $m \in \mathcal{M}$. We say Π is a reliable agreement protocol with t corruption threshold if it satisfies the following properties.*

– 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-***Totality:*** *If an honest party outputs a message, then all honest parties eventually output a message.*
– t-***Integrity:*** *If an honest party outputs $m \in \mathcal{M}$, then at least $t+1$ honest parties input m.*

Let $L \in \mathbb{N}$ and $\mathcal{M} = \{0,1\}^L \cup \{\perp\}$ denote the input and output space. In a Graded Consensus (GC) protocol, each party P_i takes $m_i \in \mathcal{M}$ as input, and it will output a pair in the form of (m_i', g_i), where $m_i' \in \mathcal{M}$ and $g_i \in \{0,1,2\}$. We recall the property-based definition of a GC protocol as follows.

Definition 5 ([FM88, DHLZ21]). *Let t be the corruption threshold, $\mathcal{M} = \{0,1\}^L \cup \{\perp\}$ be the input and output space, and $\mathcal{P}$ be the party set. Let Π be a protocol executed by parties in $\mathcal{P}$ with up to t corruptions, each party P_i takes a value $m_i \in \mathcal{M}$ as input and terminates after getting its output (m_i', g), where $m_i' \in \mathcal{M}, g \in \{0,1,2\}$. We say Π is a graded consensus protocol with t corruption threshold if it satisfies the following properties.*

– t-***Graded Validity:*** *If all honest parties have the same input $m \in \mathcal{M}$, then all honest parties take $(m,2)$ as output.*
– t-***Graded Consistency:*** *If two honest parties output grades (m,g) and (m',g'), then $|g-g'| \leq 1$. If $g, g' \geq 1$, then $m = m'$.*
– t-***Liveness:*** *Each honest parties eventually terminate with (m,g) with either $m \in \mathcal{M}, g \geq 1$, or $m = \perp, g = 0$.*
– t-***Intrusion tolerance:*** *If (m,g) is output by an honest party and $g \geq 1$, then m was input by an honest party.*

B.2 Technical Overview

In this section, we introduce our compiler to build the network-agnostic BA protocol with an optimal corruption threshold, and it also realizes the notion of "best-of-both-worlds" BA protocol defined in [ACC22, PP24].

As we mentioned before, the main bottleneck for the compiler in [ACC22, PP24] is that they use the ABA protocol with t_s corruption threshold, but the currently most efficient ABA protocol [BCP20] still incurs a large additive communication overhead of $\mathcal{O}(n^8 \log n)$ bits. To obtain a more efficient construction, we seek to avoid invoking ABA under the $t_s < n/3$ threshold.

Initial Idea and Difficulties. We first note that, when the network is asynchronous, the corruption threshold is bounded by $t_a < n/4$, and therefore an ABA protocol with t_a corruption threshold would be sufficient. A recent work [AAPS25] realizes ABA with $1/4$ corruption threshold with communication complexity

$\mathcal{O}(Ln^2 \log n + n^4 \log n)$ for L-bit agreement, which is significantly more efficient than the current best ABA with 1/3 corruption threshold [BCP20]. This motivates us to build our network-agnostic BA protocol around an ABA protocol with threshold $t_a < n/4$.

The main challenge is that ABA with threshold $t_a < n/4$ is only secure when the actual number of corrupted parties does not exceed t_a. In a synchronous execution, however, the adversary may corrupt up to $t_s > t_a$ parties. To safely execute the ABA with $t_a < n/4$ corruption threshold, we should only invoke it when the network is indeed asynchronous. Therefore, a natural idea is as follows.

- **Regular Mode**: All parties first assume the network is synchronous and try to invoke a synchronous BA (SBA) protocol with t_s corruption threshold to reach an agreement.
- **Fallback Mode**: If an honest party fails to get the output of SBA at the designed running time, which implies that the network is asynchronous, this honest party switches to executing ABA with t_a corruption threshold to reach an agreement.

Unfortunately, this natural idea does not work. Let $\mathcal{H}$ denote the set of all honest parties, and $\mathcal{H}_1$ denote the set of honest parties that terminate SBA within the designed running time. If the network is indeed synchronous, we know that $\mathcal{H}_1 = \mathcal{H}$. I.e., all honest parties will terminate SBA and never switch to the **Fallback Mode**. However, when the network is asynchronous, it is possible that some honest parties terminate SBA within the designed running time while others do not. I.e., $\mathcal{H}_1 \subsetneq \mathcal{H}$ and $\mathcal{H}_1 \neq \varnothing$. In this case, two issues arise.

- **Issue 1**. Since honest parties in $\mathcal{H}_1$ will not participate in the ABA protocol in the **Fallback Mode**, the ABA protocol may never terminate. As a result, honest parties in $\mathcal{H} \backslash \mathcal{H}_1$ may never output the agreement result.
- **Issue 2**. Even if the ABA protocol terminates, its output can be different from the output of SBA. This would result in inconsistent output between honest parties in $\mathcal{H}_1$ and those in $\mathcal{H} \backslash \mathcal{H}_1$.

Besides addressing the above two issues, which are related to the termination and consistency properties respectively, we also need to ensure the validity. In the following, we will first give our ideas to resolve the above two issues, then discuss how to guarantee the validity property.

Our Solution to Issue 1. We use a reliable agreement (RA) protocol with t_s corruption threshold to address this issue. For an RA protocol, it guarantees that: (1) once an honest party terminates, all honest parties will eventually terminate with the same results; and (2) when $n - t_s$ honest parties provide the same input, the protocol always terminates. In addition, there exists an RA protocol whose running time is deterministic in the synchronous setting.

Using an RA protocol, our solution is as follows. Parties first execute an SBA protocol and try to compute the agreement result. For parties that terminate the SBA protocol within the designed running time, they take the output of the SBA as the input for the RA protocol. Otherwise, they take a default message $\perp$ as input for the RA protocol. Then, parties check whether the RA protocol terminates within the designed running time:

- If true and the output of RA is not the default message $\perp$, they take the output of RA as the agreement result and terminate.
- Otherwise, they switch to the Fallback Mode. In addition to running the ABA protocol, they concurrently execute the RA protocol to wait for its output. Whenever a party terminates the ABA or RA with output different from $\perp$, it takes that output as the final agreement result and terminates.

Now we show that all honest parties can eventually terminate the protocol when the network is asynchronous. Let $\mathcal{H}_1$ denote instead the set of honest parties that terminate RA with output different from $\perp$ within the designed running time:

- If $\mathcal{H}_1 = \varnothing$, then all honest parties switch to execute the ABA protocol, and they will eventually get the agreement result.

– Otherwise, from the property of the RA protocol, all honest parties not in $\mathcal{H}_1$ eventually terminate the RA protocol.

When the network is synchronous, we know that all honest parties will first terminate the SBA protocol at a similar time, and therefore set the same value as input for RA. As a result, all honest parties can terminate the RA protocol with the same value after a fixed time bound.

Our Solution to Issue 2. Based on our solution to Issue 1, honest parties may terminate the protocol in two cases: receiving the RA output or the ABA output. When the network is asynchronous and honest parties have different initial inputs, the outputs from RA and ABA can be different. To achieve consistency, our idea is to let parties update their inputs to the ABA protocol such that, if any honest party terminates RA with output $y \neq \perp$ (i.e., $\mathcal{H}_1 \neq \varnothing$), then all honest parties outside $\mathcal{H}_1$ will use y as their inputs in ABA. This guarantees that honest parties will never terminate the ABA protocol with an output different from y. Note that this update is implicit: an honest party outside $\mathcal{H}_1$ may not be aware of whether the RA protocol terminates for some honest parties or not.

To realize this idea, we employ the graded consensus (GC) primitive. In a GC protocol, each honest party outputs a pair (y, g), where y is a candidate decision value and $g \in \{0, 1, 2\}$ is a confidence grade. The crucial property is that if an honest party outputs $(y, 2)$, then every honest party outputs (y, g') for some $g' \geq 1$ (graded consistency property). Moreover, if all honest parties start with the same input y, then all of them output $(y, 2)$ (graded validity property). If an honest party terminates with grade $g \geq 1$, then the corresponding candidate value is proposed by an honest party (intrusion tolerance property). There also exists a GC protocol whose running time is deterministic when the network is synchronous.

Based on the GC protocol, our construction proceeds as follows. Recall that to address Issue 1, we let parties execute the SBA and RA protocols sequentially, and now we let parties execute the GC protocol between the SBA and RA protocols. To be more concrete:

1. Every party first checks whether it has completed SBA within the designed running time. If so, it uses the SBA output as its input to GC; otherwise, it uses its original input.
2. Parties jointly invoke the GC protocol with t_s corruption threshold. Upon receiving the output of GC, parties that obtain an output in the form of (y, g) with $g \geq 1$ will update their ABA input from the original input to y.
3. Upon terminating GC, parties invoke the RA protocol where parties set y as input for RA if they receive $(y, 2)$ from the GC protocol, and $\perp$ otherwise.
4. If the RA protocol does not terminate within the designed running time, they switch to the Fallback Mode and execute the ABA protocol as our solution to issue 1.

Now, when the network is asynchronous, if honest parties in $\mathcal{H}_1$ terminate RA with output $y \neq \perp$, we know that at least one honest party receives $(y, 2)$ from the GC protocol. Based on the graded consistency property of GC, every honest party outside $\mathcal{H}_1$ must output a pair (y, g) with the same value y and grade $g \geq 1$, and will update its input for ABA to y.

Guarantee Validity. Based on the above solutions, we can guarantee the consistency and termination property of the network-agnostic BA protocol. We further check whether our solution satisfies the validity property, where we require that when all honest parties take the same value as input, they should output this common input value.

When the network is synchronous, our construction satisfies the validity property. Let x denote their common input, all honest parties will first terminate SBA with x, and then take it as the input for GC. After a fixed time bound, all of them will terminate GC with $(x, 2)$, and will set their input to RA as x. According to the property of RA, all honest parties will terminate with x within the designed running time.

However, when the network is asynchronous, the current solution is not sufficient. The reason is that when parties execute the SBA protocol under an asynchronous network, there is no security guarantee: Even if parties hold the same input x, there is no guarantee that all honest parties can output the same value. As a result, honest parties may take values different from x to GC, resulting in terminating GC with a different

candidate value $y \neq x$ and grade 1 or 2. Then, honest parties will update their ABA input to y, and they eventually terminate ABA with $y \neq x$.

To address this issue, our idea is to use an SBA protocol with the *weak validity* property in the asynchronous setting. The asynchronous *weak validity* property requires that when all honest parties take the same value x as input, none of them will terminate with a value outside $\{x, \perp\}$. Consequently, honest parties only use x or $\perp$ as input to the GC protocol. Therefore, if an honest party terminates GC with grade 1 or 2, its candidate value must be x according to the intrusion tolerance property of GC (refer to Definition 5), implying that no honest party will update its ABA input to a value different from x. Then, honest parties only input x or $\perp$ to the RA protocol, and no honest party can terminate RA with a value outside $\{x, \perp\}$. As a result, for parties that terminate with the output of RA, their output would be x. For parties that terminate with the output of ABA, their output is also x since our solutions guarantee the consistency and termination.

Summarize. The framework of our compiler is as follows. The Regular Mode includes the first three steps.

- **Step 1.** All parties invoke an SBA protocol with the *weak validity* property and use their initial inputs. During the execution of the SBA protocol with t_s corruption threshold, all parties assume the network is synchronous, and they will check whether they can terminate SBA with an output at a designed synchronous running time. If not, they set their output of SBA as their initial input.
- **Step 2.** All parties invoke a GC protocol and take their output of SBA as input. During the execution of the GC protocol with t_s corruption threshold, all parties assume the network is asynchronous, and they will wait until they receive the output of the GC protocol. The output is in the form of (y, g).
- **Step 3.** All parties invoke a RA protocol with t_s corruption threshold, and all parties assume the network is asynchronous. If a party terminates GC with $g = 2$, then it takes y as input for RA. Otherwise, it takes $\perp$ as input for RA. At a designed synchronous running time, parties will check whether they terminate RA. If true, they use the output of RA as the final output and terminate. Otherwise, they proceed to the Fallback Mode, each party that got $g \geq 1$ will take the value y it got during GC as the input for the ABA protocol during the Fallback Mode. Otherwise, it uses the original input.
- **Fallback Mode.** All parties invoke an ABA protocol with t_a corruption threshold, and all parties assume the network is asynchronous. During the execution of the ABA protocol, parties still participate in the RA protocol in Step 3. When a party terminates ABA or RA with an output different from $\perp$, it takes this output as the final agreement result and terminates.

B.3 Construction of $\Pi_{\texttt{NA-BA}}$

We use the following protocols in a black-box way:

- The synchronous Byzantine agreement protocol $\Pi_{\texttt{SBA}}$ with t_s-perfectly-secure in the synchronous setting and t_a weak validity in the asynchronous setting. We borrow the idea in [ACC22] to build such an $\Pi_{\texttt{SBA}}$, and we give the detailed construction in appendix B.4. The number of rounds for $\Pi_{\texttt{SBA}}$ is $R_{\texttt{SBA}} = 12n - 3$ when executing it in the synchronous network. The communication complexity of $\Pi_{\texttt{SBA}}$ is $\mathcal{O}(Ln^3)$ bits for L bits input.
- The graded consensus protocol $\Pi_{\texttt{GC}}$ in [BCLZL23], which achieves perfect-secure with 1/3 corrupt threshold. The number of rounds for $\Pi_{\texttt{GC}}$ is $R_{\texttt{GC}} = 10$ when executing it in the synchronous network. The communication complexity of $\Pi_{\texttt{GC}}$ is $\mathcal{O}(Ln^3)$ bits for L bits input.
- The reliable agreement $\Pi_{\texttt{RA}}$ in [Bra84], which achieves perfect-secure with 1/3 corrupt threshold. The number of rounds for $\Pi_{\texttt{RA}}$ is $R_{\texttt{RA}} = 3$ when executing it in the synchronous network and all honest parties take the same input. The communication complexity of $\Pi_{\texttt{RA}}$ is $\mathcal{O}(Ln^2)$ bits for L bits input.
- The asynchronous Byzantine agreement $\Pi_{\texttt{ABA}}$ in [AAPP24], which achieves perfect-secure with 1/4 corrupt threshold. The expected round complexity of $\Pi_{\texttt{ABA}}$ is $\mathcal{O}(1)$, and the communication complexity of $\Pi_{\texttt{ABA}}$ is $\mathcal{O}(Ln^2 \log n + n^4 \log n)$ bits in expectation for L bits input.

Protocol $\Pi_{\texttt{NA-BA}}$

Parameter: Parties agree on the following public inputs:

- The active party set $\mathcal{P}$, and we use t_s to denote the number of corrupted parties in $\mathcal{P}$ when the network is synchronous, and t_a to denote the number of corrupted parties in $\mathcal{P}$ when the network is asynchronous.
- Let Δ denote the message delay bound between any two honest parties when the network is synchronous, and let $\Delta_{\texttt{start}} < \infty$ denote the maximum difference between the starting times of any two honest parties executing the protocol under synchrony.
- Let $\Pi_{\texttt{SBA}}$ denote a synchronous BA protocol with t_s-secure in the synchronous setting and t_a-weak validity in the asynchronous setting, $\Pi_{\texttt{ABA}}$ denote a perfect-secure asynchronous BA protocol with t_a-secure, $\Pi_{\texttt{RA}}$ denote a perfect-secure reliable agreement protocol against t_s corrupted parties, and $\Pi_{\texttt{GC}}$ denote a perfect-secure asynchronous graded consensus protocol against t_s corrupted parties. We use $R_{\texttt{SBA}}, R_{\texttt{RA}}, R_{\texttt{GC}}$ to denote the number of rounds for $\Pi_{\texttt{SBA}}, \Pi_{\texttt{RA}}, \Pi_{\texttt{GC}}$ when executing them in the synchronous setting.
- Let L denote the input length, and assume each party's input is in $\{0,1\}^L$. Let $\perp$, $\perp_{\texttt{GC}}$, and $\perp_{\texttt{RA}}$ be three distinct default symbols. To distinguish these symbols from values in $\{0,1\}^L$, we encode every $x \in \{0,1\}^L$ as $(0\|x) \in \{0,1\}^{L+1}$, where the first bit is 0. For $\perp$, $\perp_{\texttt{GC}}$, and $\perp_{\texttt{RA}}$, we encode them as bitstrings in $\{0,1\}^{L+1}$ with the first bit set to 1. This ensures that all default symbols are distinguishable from valid values in $\{0,1\}^L$.

Input: Each party P_i takes a string $x^{(i)}_{\texttt{input}} \in \{0,1\}^L$ as input.

1: **Initialization.** Each party P_i initializes $(L+1)$-bit strings $x^{(i)}_{\texttt{GC}}, x^{(i)}_{\texttt{RA}}, x^{(i)}_{\texttt{ABA}}$ to be $\perp$. Party P_i also initializes a local timer $T_i = 0$.

Regular Mode

2: **Invoking Synchronous BA.** All parties invoke the following steps.

1. All parties jointly invoke an instance of $\Pi_{\texttt{SBA}}$ with corruption threshold t_s. During the execution of $\Pi_{\texttt{SBA}}$, parties assume the network is *synchronous* and set the time bound for each round of $\Pi_{\texttt{SBA}}$ to be $\Delta + \Delta_{\texttt{start}}$. Each party P_i takes $x^{(i)}_{\texttt{input}}$ as input and participates in $\Pi_{\texttt{SBA}}$ at time $T_i = 0$.
2. At time $T_i = (\Delta + \Delta_{\texttt{start}}) \cdot R_{\texttt{SBA}}$, each party P_i checks whether it has terminated $\Pi_{\texttt{SBA}}$ with output $y_{\texttt{SBA}}$.
   - If true, P_i sets $x^{(i)}_{\texttt{GC}} = (0\|y_{\texttt{SBA}})$ and proceeds.
   - Otherwise, P_i sets $x^{(i)}_{\texttt{GC}} = (0\|x^{(i)}_{\texttt{input}})$, and it *no longer* participates in $\Pi_{\texttt{SBA}}$ after $T_i > (\Delta+\Delta_{\texttt{start}}) \cdot R_{\texttt{SBA}}$.

3: **Invoking Graded Consensus.** All parties invoke the following steps.

1. All parties jointly invoke an instance of $\Pi_{\texttt{GC}}$ with corruption threshold t_s. During the execution of $\Pi_{\texttt{GC}}$, parties assume the network is *asynchronous*. Each party P_i takes its input $x^{(i)}_{\texttt{GC}}$ as input and participates in $\Pi_{\texttt{GC}}$ at time $T_i = (\Delta + \Delta_{\texttt{start}}) \cdot R_{\texttt{SBA}}$. We use $(y^{(i)}_{\texttt{GC}}, b)$ to denote the form of each P_i's output of $\Pi_{\texttt{GC}}$, where $y^{(i)}_{\texttt{GC}} \in \{0\|\{0,1\}^L\} \cup \{\perp_{\texttt{GC}}\}$ and $b \in \{0,1,2\}$.
2. Upon getting the output $(y^{(i)}_{\texttt{GC}}, b)$, if $b = 2$, P_i sets $x^{(i)}_{\texttt{RA}} = y^{(i)}_{\texttt{GC}}$. Otherwise, P_i sets $x^{(i)}_{\texttt{RA}} = \perp_{\texttt{RA}}$. Note that the output $y^{(i)}_{\texttt{GC}}$ will be $\perp_{\texttt{GC}}$ iff $b = 0$. Then if we set $x^{(i)}_{\texttt{RA}}$ as above, we guarantee that $x^{(i)}_{\texttt{RA}} \neq \perp_{\texttt{GC}}$, and is in $\{0\|\{0,1\}^L\} \cup \{\perp, \perp_{\texttt{RA}}\}$.

4: **Invoking Reliable Agreement.** All parties invoke the following steps.

1. All parties jointly invoke an instance of $\Pi_{\texttt{RA}}$ with corruption threshold t_s. During the execution of $\Pi_{\texttt{RA}}$, parties assume the network is *asynchronous*. Each party P_i participates in $\Pi_{\texttt{RA}}$ once it gets $x^{(i)}_{\texttt{RA}} \neq \perp$. We use $y_{\texttt{RA}} \in \{0\|\{0,1\}^L\} \cup \{\perp_{\texttt{RA}}\}$ to denote honest parties' output of $\Pi_{\texttt{RA}}$.
2. P_i checks whether it has terminated $\Pi_{\texttt{RA}}$ with output $y_{\texttt{RA}}$ before time $T_i = (\Delta+\Delta_{\texttt{start}}) \cdot (R_{\texttt{SBA}}+R_{\texttt{GC}}+R_{\texttt{RA}})$:
   - If true and $y_{\texttt{RA}} \neq \perp_{\texttt{RA}}$, it sets $y^{(i)}_{\texttt{output}}$ to be the last L bits of $y_{\texttt{RA}}$ and terminates with $y^{(i)}_{\texttt{output}}$.
   - Otherwise, P_i sets $x^{(i)}_{\texttt{ABA}} = y^{(i)}_{\texttt{GC}}$ if $y^{(i)}_{\texttt{GC}} \notin \{\perp, \perp_{\texttt{GC}}\}$ and $x^{(i)}_{\texttt{ABA}} = (0\|x^{(i)}_{\texttt{input}})$ otherwise. Also in this case, P_i *still* participates in $\Pi_{\texttt{RA}}$ after $T_i > (\Delta + \Delta_{\texttt{start}}) \cdot (R_{\texttt{SBA}} + R_{\texttt{GC}} + R_{\texttt{RA}})$.

Fallback Mode

5: **Fallback to Asynchronous Agreement.** All parties jointly invoke an instance of $\Pi_{\texttt{ABA}}$ with corruption threshold t_a and execute it in the asynchronous network setting. Each party P_i that has not terminated before time $T_i = (\Delta + \Delta_{\texttt{start}}) \cdot (R_{\texttt{SBA}} + R_{\texttt{GC}} + R_{\texttt{RA}})$ will participate in this $\Pi_{\texttt{ABA}}$ with its input $x^{(i)}_{\texttt{ABA}}$. Let $y_{\texttt{ABA}} \in (0\|\{0,1\}^L)$ denote the honest parties' output of $\Pi_{\texttt{ABA}}$, for each party P_i:

- If P_i first terminates $\Pi_{\texttt{RA}}$ with output $y_{\texttt{RA}} \neq \perp_{\texttt{RA}}$, it sets $y^{(i)}_{\texttt{output}}$ to be the last L bits of $y_{\texttt{RA}}$.
- If P_i first terminates $\Pi_{\texttt{ABA}}$ with output $y_{\texttt{ABA}}$, it sets $y^{(i)}_{\texttt{output}}$ to be the last L bits of $y_{\texttt{ABA}}$.

Finally, P_i terminates with $y^{(i)}_{\texttt{output}}$ as the agreement results.

Security Analysis. We first prove the consistency and termination property of Byzantine agreement.

When the network is synchronous, there are at most t_s corrupted parties, then all honest parties are guaranteed to terminate the synchronous BA protocol $\Pi_{\mathtt{SBA}}$ with t_s corruption threshold after a fixed time bound $(\Delta + \Delta_{\mathtt{start}})R_{\mathtt{SBA}}$. For all honest P_i, at time $T_i = (\Delta + \Delta_{\mathtt{start}})R_{\mathtt{SBA}}$, they get the same output $y_{\mathtt{SBA}}$ and will take the same input $x_{\mathtt{GC}}^{(i)} = (0\|y_{\mathtt{SBA}})$ for $\Pi_{\mathtt{GC}}$. Since all honest parties take the same value as input for $\Pi_{\mathtt{GC}}$, and the property of $\Pi_{\mathtt{GC}}$ guarantees that all honest parties will terminate $\Pi_{\mathtt{GC}}$ with the same output after a fixed time bound $(\Delta+\Delta_{\mathtt{start}})R_{\mathtt{GC}}$, and their output is $((0\|y_{\mathtt{SBA}}), 2)$. Therefore, each honest P_i will set $x_{\mathtt{RA}}^{(i)} = y_{\mathtt{GC}}^{(i)} = (0\|y_{\mathtt{SBA}})$. Since all honest parties take the same value $(0\|y_{\mathtt{SBA}})$ as input for $\Pi_{\mathtt{RA}}$, then the property of $\Pi_{\mathtt{RA}}$ guarantees that all honest parties will terminate $\Pi_{\mathtt{RA}}$ with the same output after a fixed time bound $(\Delta+\Delta_{\mathtt{start}})R_{\mathtt{RA}}$, and their output of $\Pi_{\mathtt{RA}}$ is $y_{\mathtt{RA}} = (0\|y_{\mathtt{SBA}})$. Upon getting the output of $\Pi_{\mathtt{RA}}$, all honest parties will terminate with the last L bits of $y_{\mathtt{RA}}$, which equals $y_{\mathtt{SBA}}$. Then we guarantee that when the network is synchronous, all honest parties will terminate $\Pi_{\mathtt{NA\text{-}BA}}$ after a fixed time bound $(\Delta+\Delta_{\mathtt{start}})(R_{\mathtt{SBA}}+R_{\mathtt{GC}}+R_{\mathtt{RA}})$, and they output the same agreement results.

When the network is asynchronous, there are at most $t_a < t_s$ corrupted parties. For the first invocation of $\Pi_{\mathtt{SBA}}$, although there is no security guarantee of honest parties' output, our construction guarantees that all honest parties will set their $x_{\mathtt{GC}}^{(i)}$ to be some value at time T_i (but maybe all honest parties have different $x_{\mathtt{GC}}^{(i)}$). Then they jointly invoke $\Pi_{\mathtt{GC}}$ with their input $x_{\mathtt{GC}}^{(i)}$ and wait until they terminate $\Pi_{\mathtt{GC}}$. Then we consider the following two cases:

- **Case 1**: At least one honest party output $(y_{\mathtt{GC}}, b)$ with $b = 2$.
- **Case 2**: No honest party output $(y_{\mathtt{GC}}^{(i)}, b)$ with $b = 2$.

We first discuss the **Case 1**. In this case, the property of $\Pi_{\mathtt{GC}}$ guarantees that the rest of the honest parties P_i will output $(y_{\mathtt{GC}}^{(i)}, b)$ with the same value $y_{\mathtt{GC}}^{(i)} = y_{\mathtt{GC}}$ and $b \geq 1$. Then they will set their $x_{\mathtt{RA}}^{(i)}$ to be $y_{\mathtt{GC}}$ or $\perp_{\mathtt{RA}}$ depending on the value of b. Note that if not all honest parties take the same value as input for $\Pi_{\mathtt{RA}}$, there is no termination guarantee of $\Pi_{\mathtt{RA}}$. But as long as one honest party terminates $\Pi_{\mathtt{RA}}$ with $y_{\mathtt{RA}}$, then we guarantee that all honest parties will eventually terminate $\Pi_{\mathtt{RA}}$ with the same output $y_{\mathtt{RA}}$. Then we further discuss two sub-cases:

- **Sub-Case 1**: At least one honest party terminate $\Pi_{\mathtt{RA}}$ with $y_{\mathtt{RA}}$ before time $(\Delta+\Delta_{\mathtt{start}})\cdot(R_{\mathtt{SBA}}+R_{\mathtt{GC}}+R_{\mathtt{RA}})$.
- **Sub-Case 2**: No honest party terminate $\Pi_{\mathtt{RA}}$ before time $(\Delta + \Delta_{\mathtt{start}}) \cdot (R_{\mathtt{SBA}} + R_{\mathtt{GC}} + R_{\mathtt{RA}})$.

For the **Sub-Case 1**, we know these honest parties that terminate $\Pi_{\mathtt{RA}}$ will directly terminate with the last L bits of $y_{\mathtt{RA}}$, and they will never participate in $\Pi_{\mathtt{ABA}}$. For parties that participate in $\Pi_{\mathtt{ABA}}$, they will set their input $x_{\mathtt{ABA}}^{(i)}$ to be $y_{\mathtt{GC}}$ since their $b \geq 1$ and $y_{\mathtt{GC}}^{(i)} = y_{\mathtt{GC}} \neq \perp_{\mathtt{GC}}$. Now we guarantee that for all honest parties:

- Honest parties that have terminated $\Pi_{\mathtt{RA}}$ with $y_{\mathtt{RA}}$ will terminate the protocol with the last L bits of $y_{\mathtt{RA}}$, and these honest parties will not participate in $\Pi_{\mathtt{ABA}}$.
- Honest parties that participate in $\Pi_{\mathtt{ABA}}$ will take the same value $x_{\mathtt{ABA}}^{(i)} = y_{\mathtt{RA}}$ as input.

Based on the validity property of $\Pi_{\mathtt{ABA}}$, we know that no honest parties will terminate $\Pi_{\mathtt{ABA}}$ with output $y_{\mathtt{ABA}} \neq y_{\mathtt{RA}}$ since no honest party take $x_{\mathtt{ABA}}^{(i)} \neq y_{\mathtt{RA}}$ as input. Although there is no termination guarantee of $\Pi_{\mathtt{ABA}}$ since not all honest parties participate in $\Pi_{\mathtt{ABA}}$, the property of RA guarantees that all honest parties will eventually terminate with the same $y_{\mathtt{RA}}$, and therefore get the same agreement results.

For the **Sub-Case 2**, all honest parties will participate in $\Pi_{\mathtt{ABA}}$ with their input $x_{\mathtt{ABA}}^{(i)}$, and eventually terminate $\Pi_{\mathtt{ABA}}$ with the same results.

Then we discuss the **Case 2**. In this case, all honest P_i will set their $x_{\mathtt{RA}}^{(i)}$ to be $\perp_{\mathtt{RA}}$, and the property of $\Pi_{\mathtt{RA}}$ guarantees that all honest parties will eventually terminate $\Pi_{\mathtt{RA}}$ with $y_{\mathtt{RA}} = \perp_{\mathtt{RA}}$. Therefore, no honest parties will terminate the protocol after $\Pi_{\mathtt{RA}}$, they will take their $x_{\mathtt{ABA}}^{(i)}$ as input for $\Pi_{\mathtt{ABA}}$, and eventually terminate $\Pi_{\mathtt{ABA}}$ with the same agreement results.

Now we discuss the validity property of our BA construction.

When the network is synchronous, according to the validity property of the synchronous BA $\Pi_{\mathtt{SBA}}$, when all honest parties take the same value as input x, they will terminate $\Pi_{\mathtt{SBA}}$ with the same results $y_{\mathtt{SBA}} = x$. As we analyzed above, all of them will eventually terminate with $y_{\mathtt{SBA}} = x$.

When the network is asynchronous, according to the t_a-Async weak validity property of $\Pi_{\mathtt{SBA}}$, we know that no honest parties will terminate $\Pi_{\mathtt{SBA}}$ with $y_{\mathtt{SBA}} \neq x$ at time T_i. Therefore, all honest parties will take the same value $(0\|x)$ as input for $\Pi_{\mathtt{GC}}$. Then, according to the graded validity property of $\Pi_{\mathtt{GC}}$, all honest parties will eventually terminate $\Pi_{\mathtt{GC}}$ with $((0\|x), 2)$. Then they will set the same value $(0\|x)$ as input for $\Pi_{\mathtt{RA}}$, and the property of $\Pi_{\mathtt{RA}}$ guarantees that for parties that terminate $\Pi_{\mathtt{RA}}$, their output is the same $(0\|x)$. Parties that participate in $\Pi_{\mathtt{ABA}}$ also take the value $(0\|x)$ as input, and honest parties either wait for the termination of $\Pi_{\mathtt{RA}}$ to get $(0\|x)$, or eventually terminate $\Pi_{\mathtt{ABA}}$ with $(0\|x)$. Therefore, all honest parties also obtain the same agreement results, x, when the network is asynchronous.

Communication and Round Complexity. All parties invoke the protocol $\Pi_{\mathtt{SBA}}, \Pi_{\mathtt{GC}}, \Pi_{\mathtt{RA}}$, and $\Pi_{\mathtt{ABA}}$ in sequence. Note that when the network is synchronous. all parties are guaranteed to terminate at the end of $\Pi_{\mathtt{RA}}$, and the total communication complexity is $\mathcal{O}(Ln^3)$ bits for L-bit agreement. When the network is asynchronous, parties may proceed to execute the $\Pi_{\mathtt{ABA}}$, which requires additional $\mathcal{O}(Ln^2 \log n + n^4 \log n)$ bits of communication in expectation for L-bit agreement. For simplicity, we consider the overall communication complexity of $\Pi_{\mathtt{NA\text{-}BA}}$ is $\mathcal{O}(Ln^3 + n^4 \log n)$ bits in expectation for L-bit agreement.

For the round complexity, when the network is synchronous, it costs $\mathcal{O}(n)$ rounds for honest parties to terminate $\Pi_{\mathtt{SBA}}$ and $\mathcal{O}(1)$ rounds for honest parties to terminate $\Pi_{\mathtt{GC}}$ and $\Pi_{\mathtt{RA}}$, then the round complexity is $\mathcal{O}(n)$.

When the network is asynchronous, we use the definition of round complexity in [CGHZ16]. In [CGHZ16], a party can be in two states:(1) sending messages, or (2) waiting for messages, and the round complexity is the number of switches between these two states. Based on this definition, the round complexity of $\Pi_{\mathtt{SBA}}$ is still $\mathcal{O}(n)$, and the round complexity of $\Pi_{\mathtt{GC}}, \Pi_{\mathtt{RA}}$ is still $\mathcal{O}(1)$. Since parties may execute $\Pi_{\mathtt{ABA}}$ in the asynchronous setting, it requires additional $\mathcal{O}(1)$ rounds in expectation to terminate.

B.4 Construction of Synchronous BA with additional Weak Validity in the Asynchronous Setting

In this section, we introduce the idea of building the SBA protocol with the t_a weak validity property in the asynchronous setting. This idea is borrowed from [ACC22, PP24]. To build $\Pi_{\mathtt{SBA}}$ with this addition property, we require the following building blocks:

- A reliable broadcast protocol with t_s corruption threshold. We can use Bracha's reliable broadcast protocol $\Pi_{\mathtt{Acast}}$ [Bra84] with t_s corruption threshold. The number of rounds for $\Pi_{\mathtt{Acast}}$ is $R_{\mathtt{Acast}} = 3$ when executing it in the synchronous network. The communication complexity of $\Pi_{\mathtt{Acast}}$ is $\mathcal{O}(n^2)$ bits for one-bit input.
- A synchronous BA protocol with t_s-secure. We can use the construction in [BGP92], denoted by $\Pi'_{\mathtt{SBA}}$. The number of rounds for $\Pi'_{\mathtt{SBA}}$ is $R_{\mathtt{SBA}'} = 12n - 6$ when executing it in the synchronous network. The communication complexity of $\Pi'_{\mathtt{SBA}}$ is $\mathcal{O}(n^2)$ bits for one-bit input.

Protocol $\Pi_{\mathtt{SBA}}$

Parameter: Parties agree on the following public inputs:
- The active party set $\mathcal{P}$, and t_s denote the number of corrupted parties in $\mathcal{P}$.
- Let Δ denote the message delay bound between any two honest parties when the network is synchronous, and let $\Delta_{\mathtt{start}} < \infty$ denote the maximum difference between the starting times of any two honest parties executing the protocol under synchrony.
- Let $\Pi'_{\mathtt{SBA}}$ denote a synchronous BA protocol with t_s-secure in the synchronous setting and $\Pi_{\mathtt{Acast}}$ to denote an reliable broadcast protocol with t_s corruption threshold. We use $R_{\mathtt{SBA}'}, R_{\mathtt{Acast}}$ to denote the number of rounds for $\Pi'_{\mathtt{SBA}}, \Pi_{\mathtt{Acast}}$ when executing them in the synchronous setting.

Input: Each party P_i holds an input bit $b_i \in \{0, 1\}$, and initializes a set $\mathcal{W}_i = \varnothing$, timer $T_i = 0$.

1: Each party P_i acts as the dealer and leads an instance of $\Pi_{\texttt{Acast}}$, denoted by $\Pi^{(i)}_{\texttt{Acast}}$. The party P_i takes b_i as input for $\Pi^{(i)}_{\texttt{Acast}}$, and each party P_k participates in $\Pi^{(i)}_{\texttt{Acast}}$ at time $T_k = 0$. During the execution of each $\Pi^{(i)}_{\texttt{Acast}}$, parties assume the network is *synchronous* and set the time bound for each round of $\Pi^{(i)}_{\texttt{Acast}}$ to be $\Delta + \Delta_{\texttt{start}}$.

2: At time $T_i = (\Delta + \Delta_{\texttt{start}}) \cdot R_{\texttt{Acast}}$, for each sender P_j, each party P_i participates in an instance of $\Pi'^{(j)}_{\texttt{SBA}}$ with its input set as follows:
   - If $m'_j \in \{0, 1\}$ is received from $\Pi^{(j)}_{\texttt{Acast}}$, then P_i sets m'_j as the input.
   - Otherwise, P_i sets its input as $\perp$.

3: At time $T_i = (\Delta + \Delta_{\texttt{start}}) \cdot (R_{\texttt{Acast}} + R_{\texttt{SBA}'})$, for each sender P_j, each P_i computes a value $x^{(i)}_j$ as follows:
   - If $m'_j \in \{0, 1\}$ is received from $\Pi^{(j)}_{\texttt{Acast}}$ and m'_j is computed as the output of $\Pi'^{(j)}_{\texttt{SBA}}$, then P_i sets $x^{(i)}_j = m'$.
   - Otherwise, P_i sets its input as $x^{(i)}_j = \perp$.

4: At time $T_i = (\Delta + \Delta_{\texttt{start}}) \cdot (R_{\texttt{Acast}} + R_{\texttt{SBA}'})$, for all $j \in [n]$, P_i includes P_j to $\mathcal{W}_i$ if $x^{(i)}_j \neq \perp$. Then P_i computes its final output $y^{(i)}$ as follows.
   - If $|\mathcal{W}_i| \geq n - t_s$, then set $y^{(i)}$ to the majority bit among the $x^{(i)}_j$ values of the parties in $\mathcal{W}_i$. If there is no majority, then set $y^{(i)} = 1$.
   - Otherwise, set $y^{(i)} = \perp$.

   Each party P_i terminates with $y^{(i)}$.

The total communication complexity of $\Pi_{\texttt{SBA}}$ is $\mathcal{O}(n^3)$ bits for one-bit agreement, and it requires $12n - 3$ rounds to terminate when the network is synchronous. In the following, we prove that the above protocol $\Pi_{\texttt{SBA}}$ is a t_s-perfectly-secure synchronous BA protocol with $t_s > t_a$ asynchronous weak validity.

We first prove that it is a t_s-perfectly-secure synchronous BA protocol.

For t_s-Validity: When the network is synchronous, and all honest parties hold the same input x, we know that each honest party will receive $m'_j = x$ from $n - t_s$ distinct honest P_j at time $T_i = (\Delta + \Delta_{\texttt{start}}) \cdot (R_{\texttt{Acast}} + R_{\texttt{SBA}'})$. Since $n - t_s > t_s$, then the value x becomes the majority in $\mathcal{W}_i$, and $|\mathcal{W}_i| \geq n - t_s$. Then each honest party P_i will output the value x, which is the honest parties' common input.

For t_s-Consistency: When the network is synchronous, we show that each pair of honest parties (P_i, P_j) holds the same set $\mathcal{M}_i = \mathcal{M}_j$, and therefore their output is consistent. This is because all honest parties execute an instance of $\Pi'^{(j)}_{\texttt{SBA}}$ for each party P_j, then all honest parties will receive the same value from $\Pi'^{(j)}_{\texttt{SBA}}$ at time $T_i = (\Delta + \Delta_{\texttt{start}}) \cdot (R_{\texttt{Acast}} + R_{\texttt{SBA}'})$, no matter P_j is honest or corrupted.

For t_s-Termination: When the network is synchronous, the running time of sub-protocols $\Pi_{\texttt{Acast}}$ and $\Pi'_{\texttt{SBA}}$ is deterministic, and all honest parties will terminate the protocol after a fixed time bound.

Then we show that it satisfies the t_s asynchronous weak validity. We consider the case when the network is asynchronous, and all honest parties take the same input value x. For each dealer P_j, it broadcasts the a value by $\Pi^{(j)}_{\texttt{Acast}}$. Then at the time $T_i = (\Delta + \Delta_{\texttt{start}}) \cdot R_{\texttt{Acast}}$, each P_i will set its input for $\Pi'^{(j)}_{\texttt{SBA}}$ according to whether it has received the broadcast value from P_j. As a result, for each honest dealer P_j, all honest parties either take x or $\perp$ as input for $\Pi'^{(j)}_{\texttt{SBA}}$, and they will terminate $\Pi'^{(j)}_{\texttt{SBA}}$ within a fixed time bound. Then they check whether their output of $\Pi^{(j)}_{\texttt{Acast}}$ and $\Pi'^{(j)}_{\texttt{SBA}}$ is the same. For each honest P_j, we know that they broadcast the common value x by $\Pi^{(j)}_{\texttt{Acast}}$, and therefore no honest party will set $x^{(i)}_j$ to be a value not in $\{x, \perp\}$. As a result, if an honest party P_i gets $\mathcal{W}_i$ with $|\mathcal{W}_i| \geq n - t_s$ at time $T_i = (\Delta + \Delta_{\texttt{start}}) \cdot (R_{\texttt{Acast}} + R_{\texttt{SBA}'})$, we know that it includes at least $n - t_s - t_s \geq t_s + 1 > t_s$ distinct honest P_j, and P_i receives the same value x from these honest parties. Therefore, x will be the majority value, and no honest party will output a value not in $\{x, \perp\}$.