Revisiting the Efficiency of Asynchronous MPC with Optimal Resilience Against General Adversaries*

Ananya Appan[†] Anirudh Chanramouli[‡] Ashish Choudhury[§]

Abstract

In this paper, we design *unconditionally-secure* multi-party computation (MPC) protocols in the *asynchronous* communication setting with *optimal* resilience. Our protocols are secure against a computationally unbounded *malicious* adversary characterized by an *adversary structure* $\mathcal{Z}$, which enumerates all possible subsets of potentially corrupt parties. We present protocols with both *perfect-security*, as well as with *statistical-security*. While the protocols in the former class achieve all the security properties in an error-free fashion, the protocols belonging to the latter category achieve all the security properties except with a negligible error.

Our perfectly-secure protocol incurs an *amortized* communication of $\mathcal{O}(|\mathcal{Z}|^2)$ bits per multiplication. This improves upon the protocol of Choudhury and Pappu (INDOCRYPT 2020) with the least known *amortized* communication complexity of $\mathcal{O}(|\mathcal{Z}|^3)$ bits per multiplication. On the other hand, our statistically-secure protocol incurs an *amortized* communication of $\mathcal{O}(|\mathcal{Z}|)$ bits per multiplication. This is the first statistically-secure asynchronous MPC protocol against general adversaries. Previously, perfectly-secure and statistically-secure MPC with an amortized communication cost of $\mathcal{O}(|\mathcal{Z}|^2)$ and $\mathcal{O}(|\mathcal{Z}|)$ bits respectively per multiplication were known *only* in the relatively simpler *synchronous* communication setting (Hirt and Tschudi, ASIACRYPT 2013).

1 Introduction

Secure *multi-party computation* (MPC) [34, 20, 5, 32] is a fundamental problem in secure distributed computing. Consider a set of n mutually-distrusting parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, where a subset of parties can be corrupted by a *computationally-unbounded malicious* (Byzantine) adversary Adv. Informally, an MPC protocol allows the parties to securely compute any function f of their private inputs, while ensuring that their respective inputs remain private. The most popular way of characterizing Adv is through a *threshold*, by assuming that it can corrupt *any* subset of up to t parties. In this setting, MPC with *perfect security* (where no error is allowed in the security properties) is achievable if and only if $t < n/3$ [5]. On the other hand, if a negligible error is allowed in the security properties, then one can tolerate up to $t < n/2$ corruptions [32]. Protocols of the latter class are *statistically-secure*.

Hirt and Maurer [23] generalized the threshold model by introducing the general-adversary model (also known as the *non-threshold* setting). In this setting, Adv is characterized by a *monotone adversary structure*

*A preliminary version of this article is published as an extended abstract at INDOCRYPT 2022; this is the full and elaborate version with full security proofs, along with the results for statistically secure MPC.

[†]The work was done as a student at the International Institute of Information Technology, Bangalore

[‡]Department of Computer Science, Bar-Ilan University. The work was done as a student at the International Institute of Information Technology, Bangalore

[§]International Institute of Information Technology, Bangalore India. Email: `ashish.choudhury@iiitb.ac.in`. This research is an outcome of the R&D work undertaken in the project under the Visvesvaraya PhD Scheme of Ministry of Electronics & Information Technology, Government of India, being implemented by Digital India Corporation (formerly Media Lab Asia). The author is also thankful to the Electronics, IT & BT Government of Karnataka for supporting this work under the CIET project.

$\mathcal{Z} = \{Z_1, \ldots, Z_h\} \subset 2^{\mathcal{P}}$, which enumerates all possible subsets of potentially corrupt parties, where Adv can select any subset of parties $Z^\star$ for corruption, where $Z^\star \in \mathcal{Z}$. The monotonicity of the adversary structure here implies that if some $Z \in \mathcal{Z}$, then every subset of Z also belongs to $\mathcal{Z}$. Modelling the distrust in the system through $\mathcal{Z}$ allows for more flexibility (compared to the threshold model), especially when $\mathcal{P}$ is not too large. Following the terminology of [23], given $\mathcal{P}' \subseteq \mathcal{P}$, we say that $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(k)}(\mathcal{P}', \mathcal{Z})$ condition, if the union of *any* k subsets from $\mathcal{Z}$, *does not* cover the entire set of parties in $\mathcal{P}'$. That is, if for every $Z_{i_1}, \ldots, Z_{i_k} \in \mathcal{Z}$, the following condition holds:

$$\mathcal{P}' \not\subseteq Z_{i_1} \cup \ldots \cup Z_{i_k}.$$

In the general-adversary model, MPC with perfect security is achievable if and only if $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition, while statistical security is achievable if and only if $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(2)}(\mathcal{P}, \mathcal{Z})$ condition [23, 28].

Following the seminal work of [5] all generic perfectly-secure MPC protocols follow the paradigm of *shared circuit-evaluation*. In this paradigm, it is assumed that f is abstracted as a publicly-known arithmetic circuit ckt over some finite field $\mathbb{F}$. The problem of securely computing f then boils down to "securely evaluating" the circuit ckt. To achieve this goal, the parties jointly and securely evaluate each gate in ckt in a secret-shared fashion, where each value during the circuit-evaluation remains secret-shared. In more detail, each party first secret-shares its input for f, with every party holding a share for each input, such that the shares of the corrupt parties reveal no additional information about the underlying shared values. The parties then maintain the following *gate-invariant* for each gate in ckt:

Given the gate-inputs in a secret-shared fashion, the parties get the gate-output in a secret-shared fashion without revealing any additional information about the gate-inputs and gate-output.

Finally, the function-output (which is secret-shared) is publicly reconstructed. Intuitively, security follows because the adversary does not learn any additional information beyond the inputs of the corrupt parties and the function output, since the shares learnt by Adv are independent of the underlying values.

How the above gate-invariant is maintained depends on the type of gate and the type of secret-sharing used. Typically, the underlying secret-sharing is *linear*, such as Shamir's [33] for the case of *threshold adversaries*, and replicated secret-sharing [25, 28] for the case of general adversaries.[1] Consequently, maintaining the invariant for linear gates in ckt is "free" (completely *non-interactive*). However, to maintain the gate-invariant for non-linear (multiplication) gates, the parties need to interact. Consequently, the communication complexity (namely, the total number of bits communicated by uncorrupted parties) of any generic MPC protocol is dominated by the communication complexity of evaluating the multiplication gates in ckt. Hence, the focus is to improve the *amortized* communication complexity per multiplication gate. The amortized complexity is derived under the assumption that the circuit is "large enough", so that the terms that are *independent* of the circuit size can be ignored.

In terms of communication efficiency, MPC protocols against general adversaries are *inherently less efficient* than those against threshold adversaries, by several orders of magnitude. Protocols against threshold adversaries typically incur an *amortized* communication of $n^{\mathcal{O}(1)}$ bits per multiplication, compared to $|\mathcal{Z}|^{\mathcal{O}(1)}$ bits per multiplication required against general adversaries. Since $|\mathcal{Z}|$ could be exponentially large in n, the *exact* exponent is very important. For instance, as noted in [24], if $n = 25$, then $|\mathcal{Z}|$ is around one million. Consequently, a protocol with an amortized communication complexity of $\mathcal{O}(|\mathcal{Z}|^2)$ bits per multiplication is preferred over a protocol with an amortized communication complexity of $\mathcal{O}(|\mathcal{Z}|^3)$ bits. The most efficient perfectly-secure MPC protocol against general adversaries is due to [24], which incurs

[1] A secret-sharing scheme is called *linear*, if the shares are computed as a linear function of the secret and the underlying randomness used in the scheme.

an *amortized* communication of $\mathcal{O}(|\mathcal{Z}|^2 \cdot (n^5 \log |\mathbb{F}| + n^6) + |\mathcal{Z}| \cdot (n^7 \log |\mathbb{F}| + n^8))$ bits per multiplication.[2] The work of [24] also presents the most efficient statistically-secure MPC protocol, with an amortized communication of $\mathcal{O}(|\mathcal{Z}| \cdot n^5 \log |\mathbb{F}|)$ bits per multiplication.

Our Motivation and Results. All the above results hold in the *synchronous* setting, where the parties are assumed to be globally synchronized, with strict upper bounds on the message delay. Hence, any "late" message is attributed to a corrupt sender party. Such strict time-outs are, however, extremely difficult to maintain in real-world networks like the Internet, which are better modelled by the *asynchronous* communication setting [7]. Here, no timing assumptions are made and messages can be arbitrarily (but finitely) delayed, with every message sent being delivered *eventually*. Furthermore, messages need not be delivered in the order in which they were sent. Apart from modelling real-world networks better, asynchronous protocols have the advantage of running at the "actual speed" of the underlying network. More specifically, in a synchronous protocol, the participants have to pessimistically set the global message delay, say Δ, to a large value, to ensure that all the messages sent by the different parties at the beginning of a round are delivered within time Δ. However, if the actual message delay of the network δ is such that $\delta << \Delta$, then the protocol fails to take advantage of the faster network and its running time will still be proportional to Δ.

Unfortunately, asynchronous protocols are inherently more complex and less efficient by several orders of magnitude when compared to their synchronous counterparts. This is because, in any asynchronous protocol, a *slow* (but uncorrupted) sender party *cannot* be distinguished from a *corrupt* sender party who does not send any message. Consequently, to avoid an endless wait, the parties *cannot* afford to wait to receive messages from all the parties, which results in unknowingly ignoring messages from a subset of potentially honest parties. The *resilience* (fault-tolerance) of *asynchronous* MPC (AMPC) protocols is poor compared to synchronous MPC protocols. For instance, perfectly-secure AMPC against threshold adversaries is achievable if and only if $t < n/4$ [4], while statistically-secure AMPC is achievable if and only if $t < n/3$ [6, 1]. Against general adversaries, perfectly-secure and statistically-secure AMPC require $\mathcal{Z}$ to satisfy the $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ and $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ conditions respectively.

Although more practical when compared to synchronous MPC protocols, AMPC protocols are not very well-studied [4, 6, 3, 30, 13], especially against general adversaries. To the best of our knowledge, the most efficient perfectly-secure AMPC protocol against general adversaries is due to [12], which incurs an *amortized* communication of $\mathcal{O}(|\mathcal{Z}|^3 \cdot (n^7 \log |\mathbb{F}| + n^9 \cdot (\log n + \log |\mathcal{Z}|)))$ bits per multiplication. On the other hand, there exists *no* statistically-secure AMPC protocol against general adversaries. In [12], it was left as an open problem to further improve the (amortized) communication complexity of their protocol and to bridge the efficiency gap between the communication complexity of synchronous and asynchronous MPC protocols against general adversaries. In this work, we make efforts towards solving this problem by presenting novel perfectly-secure and statistically-secure AMPC protocols against general adversaries. The *amortized* communication complexities of these protocols per multiplication are $\mathcal{O}(|\mathcal{Z}|^2 \cdot n^7 \log |\mathbb{F}| + |\mathcal{Z}| \cdot n^9 \log n)$ and $\mathcal{O}(|\mathcal{Z}| \cdot n^9 \log |\mathbb{F}|)$ bits respectively. The *amortized* efficiency of our protocols are comparable with the most efficient perfectly-secure and statistically-secure MPC protocols against general adversaries in the *synchronous* communication setting [24], especially if we focus on the exponent of $|\mathcal{Z}|$. Our results compared with relevant existing results are presented in Table 1.

1.1 Technical Overview

Our protocol is designed in the pre-processing model, where the parties first generate secret-shared random *multiplication-triples* over $\mathbb{F}$ of the form (a, b, c), where $c = ab$. The triples can be generated in a *function-independent* per-processing phase. The triples are used later to efficiently evaluate the multiplication gates

[2] The complexity is derived by substituting the broadcasts done in their protocol through the reliable broadcast protocol of [18], as referred in [24].

Setting	Security	Necessary Condition	Reference	Communication Complexity
Synchronous	Perfect	$\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$	[24]	$\mathcal{O}(\|\mathcal{Z}\|^2 \cdot (n^5 \log \|\mathbb{F}\| + n^6) + \|\mathcal{Z}\| \cdot (n^7 \log \|\mathbb{F}\| + n^8))$
Synchronous	Statistical	$\mathbb{Q}^{(2)}(\mathcal{P}, \mathcal{Z})$	[24]	$\mathcal{O}(\|\mathcal{Z}\| \cdot n^5 \log \|\mathbb{F}\|)$
Asynchronous	Perfect	$\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$	[12]	$\mathcal{O}(\|\mathcal{Z}\|^3 \cdot (n^7 \log \|\mathbb{F}\| + n^9 \cdot (\log n + \log \|\mathcal{Z}\|)))$
Asynchronous	Perfect	$\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$	This work	$\mathcal{O}(\|\mathcal{Z}\|^2 \cdot n^7 \log \|\mathbb{F}\| + \|\mathcal{Z}\| \cdot n^9 \log n)$
Asynchronous	Statistical	$\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$	This work	$\mathcal{O}(\|\mathcal{Z}\| \cdot n^9 \log \|\mathbb{F}\|)$

Table 1: Amortized communication complexity per multiplication of different MPC protocols against general adversaries

in ckt using Beaver's method [2]. At the heart of our pre-processing phase lies efficient *asynchronous* multiplication protocols to securely multiply two secret-shared values. The protocols closely follow the *synchronous* multiplication protocols of [24]. However, there are several non-trivial challenges (discussed in the sequel) while adapting these protocols to the *asynchronous* setting.

1.1.1 Perfectly-Secure Multiplication Protocol of [24] and Challenges in the Asynchronous Setting

The perfectly-secure MPC protocol of [24] as well as ours is based on the secret-sharing used in [28]. The secret-sharing is based on a given *sharing specification* $\mathbb{S}$, which is a collection of the *complement* of every subset from the underlying adversary structure $\mathcal{Z}$. That is, given $\mathcal{Z}$, the corresponding sharing specification is defined as

$$\mathbb{S} \stackrel{def}{=} \{S_q = \mathcal{P} \setminus Z_q | Z_q \in \mathcal{Z}\}.$$

Then a value $s \in \mathbb{F}$ is said to be secret-shared with respect to $\mathbb{S}$ if both the following holds.

– There exist *shares* $s_1, \ldots, s_{|\mathbb{S}|} \in \mathbb{F}$ such that $s = s_1 + \ldots + s_{|\mathbb{S}|}$;

– For $q = 1, \ldots, |\mathbb{S}|$, the share s_q is known to every (honest) party in S_q.

A secret-sharing of s as above is denoted by $[s]$, where $[s]_q$ denotes the q^{th} share. Note that if the shares $[s]_q$ are selected *randomly* from $\mathbb{F}$, then the probability distribution of the shares learnt by the adversary will be *independent* of s. This is because the set $\mathbb{S}$ consists of at least one subset, which *excludes* all the *corrupt* parties among $\mathcal{P}$, such that the corresponding share (which is selected randomly) is *unavailable* with the adversary. We now describe the perfectly-secure *synchronous* multiplication protocol of [24].

The Perfectly-Secure Multiplication Protocol of [24]. The protocol assumes that the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition holds, and takes as input secret-sharings $[a], [b]$ of a and b respectively to securely generate a random sharing $[ab]$. Note that the following holds:

$$ab = \sum_{(p,q) \in \{1,\ldots,|\mathbb{S}|\} \times \{1,\ldots,|\mathbb{S}|\}} [a]_p [b]_q.$$

The main idea is that since $S_p \cap S_q \neq \emptyset$, a *publicly-known* party from $S_p \cap S_q$ can be *designated* to secret-share the summand $[a]_p[b]_q$. For efficiency, every designated "summand-sharing party" can sum up all the summands assigned to it and share the sum instead. If *no* summand-sharing party behaves *maliciously*, then the sum of all secret-shared sums leads to a secret-sharing of ab.

To deal with maliciously-corrupt summand-sharing parties, [24] first designed an *optimistic* multiplication protocol Π_{OptMult}, which takes an additional parameter $Z \in \mathcal{Z}$ and generates a secret-sharing of ab, *provided* Adv corrupts a set of parties $Z^\star \subseteq Z$. The idea used in Π_{OptMult} is the same as above, except that the summand-sharing parties are now "restricted" to the subset $\mathcal{P} \setminus Z$. Since $(S_p \cap S_q) \setminus Z$ will be non-empty (as otherwise $\mathcal{Z}$ *does not* satisfy the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition), it is guaranteed that each summand $[a]_p[b]_q$ can be assigned to a designated party in $\mathcal{P} \setminus Z$. Since the parties will *not* know the exact set of corrupt parties, they run an instance of Π_{OptMult}, once for each $Z \in \mathcal{Z}$. This guarantees that at least one of these instances generates a secret-sharing of ab. By comparing the output sharings generated in all the instances of Π_{OptMult}, the parties can *detect* whether any cheating has occurred. If no cheating is detected, then any of the output sharings can serve as the sharing of ab. Else, the parties consider a pair of *conflicting* Π_{OptMult} instances (whose resultant output sharings are different) and proceed to a *cheater-identification* phase. In this phase, based on the values shared by the summand-sharing parties in the conflicting Π_{OptMult} instances, the parties identify at least one corrupt summand-sharing party. This phase *necessarily* requires the participation of *all* the summand-sharing parties from the conflicting Π_{OptMult} instances. Once a corrupt summand-sharing party is identified, the parties disregard all output sharings of Π_{OptMult} instances involving that party. This process of comparing the output sharings of Π_{OptMult} instances and identifying corrupt parties continues, until all the remaining output sharings are for the same value.

Challenges in the Asynchronous Setting. There are *two* main non-trivial challenges while applying the above ideas in an *asynchronous* setting. First, in Π_{OptMult}, a potentially *corrupt* party may *never* share the sum of the summands *designated* to that party, leading to an *indefinite* wait. To deal with this, we notice that since $\mathcal{Z}$ will now be satisfying the $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition (since we are in the *asynchronous* world), each $(S_p \cap S_q) \setminus Z$ contains at least one *honest* party. So instead of designating a *single* party for the summand $[a]_p[b]_q$, *each* party in $\mathcal{P} \setminus Z$ shares the sum of *all* the summands it is "capable" of, thus guaranteeing that each $[a]_p[b]_q$ is considered for sharing by at least one (honest) party. A special care is taken to ensure that a summand is *not* shared multiple times.

The *second* challenge is that once the parties identify a pair of conflicting Π_{OptMult} instances, the potentially *corrupt* summand-sharing parties from these instances *may not* participate further in the cheater-identification phase, leading to an *indefinite* wait. To get around this problem, the multiplication protocol proceeds in *iterations*, where in each iteration, the parties run an instance of the *asynchronous* Π_{OptMult} (outlined above) for each $Z \in \mathcal{Z}$ and then compare the outputs from each instance. They then proceed to the respective cheater-identification phase if the outputs are not the same. However, the summand-sharing parties from previous iterations are *not* allowed to participate in future iterations until they participate in the cheater-identification phase of all the previous iterations. This prevents the *corrupt* summand-sharing parties in previous iterations from acting as summand-sharing parties in future iterations, until they clear their "pending tasks", in which case they are caught and discarded forever. We stress that the *honest* summand-sharing parties from pending cheater-identification phases are eventually "released", so that they can act as summand-sharing parties in future iterations. Thus, even if the corrupt summand-sharing parties from previous iterations are "stuck" forever, the parties eventually progress to the next iteration, if the current iteration "fails". Once the parties reach an iteration where the outputs of all the Π_{OptMult} instances are the same (which happens eventually), the protocol stops.

Even though the above modifications (for the asynchronous setting) might look trivial, realizing them is highly challenging and technical. Hence, we defer to Section 3 for a more detailed overview and formal details.

1.1.2 Statistically-Secure Multiplication Protocol of [24] and Extension in the Asynchronous Setting

Since, in the (synchronous) statistical setting, $\mathcal{Z}$ satisfies *only* $\mathbb{Q}^{(2)}(\mathcal{P}, \mathcal{Z})$ condition, reconstructing a value s which is secret-shared with respect to the sharing specification $\mathbb{S} \stackrel{def}{=} \{\mathcal{P} \setminus Z | Z \in \mathcal{Z}\}$ *may not* be robust. This is because there may not be sufficiently many honest parties in the sets $S_q \in \mathbb{S}$ and hence the parties may fail to get the correct share $[s]_q$. To get rid of this problem in the statistical setting, [24] "enhances" the secret-sharing by ensuring that the share held by each party in S_q is "authenticated" by every other party in S_q. The authentication mechanism is achieved by deploying unconditionally-secure *Information-Checking*

(IC) signatures [32], for which an *information-checking protocol* (ICP) tolerating general adversaries is presented in [24]. They then present an optimistic multiplication protocol, which takes as input enhanced secret-sharings $[a], [b]$ of a and b, and outputs an enhanced secret-sharing of $[ab]$ *provided* no cheating occurs. The idea behind the optimistic multiplication is similar to the perfectly-secure protocol, where each summand $[a]_p[b]_q$ is designated to a *single* party in $S_p \cap S_q$ (since $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(2)}(\mathcal{P}, \mathcal{Z})$ condition, $S_p \cap S_q \neq \emptyset$), who secret-shares $[a]_p[b]_q$. Once again for efficiency, each summand-sharing party secret-shares the sum of all the summands designated to it.

Note that *unlike* the case of perfect security, the optimistic multiplication protocol *does not* take into account any subset $Z \in \mathcal{Z}$. This is because $(S_p \cap S_q) \setminus Z$ may be empty and consequently, there may not be any party outside the set Z which can be designated to secret-share the summand $[a]_p[b]_q$. Thus, *instead* of running $|\mathbb{S}|$ instances of the optimistic multiplication protocol (one instance corresponding to every $Z \in \mathbb{S}$), *only* a single instance is executed. However, if the corrupt parties behave maliciously during the instance, then the end result will not be a secret-sharing of ab. To detect any cheating that may have occurred, the parties deploy probabilistic checks. If any cheating is detected, the parties publicly identify at least one corrupt party by reconstructing all the values involved in the protocol. The identified corrupt party(ies) are discarded and the whole process is repeated again, till no cheating is detected. Note that in case the process is repeated, then values a and b from failed instances are not considered towards generating the shared random multiplication-triples. Any subsequent instance of the optimistic multiplication protocol runs with fresh, independent secret-shared a and b values jointly generated by the parties.

Extension in the Asynchronous Setting. To the best of our knowledge, no prior work has presented a statistically-secure AMPC protocol against general adversaries. Thus, to extend the above ideas to the asynchronous setting (where $\mathcal{Z}$ now satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition), we have to start from "scratch". We first design an *asynchronous information-checking protocol* (AICP) against general adversaries. Equipped with this AICP, we then design a statistically-secure *asynchronous verifiable secret-sharing* (AVSS) scheme. The AVSS protocol allows us to generate a secret-sharing of a value with respect to the sharing specification $\mathbb{S}$ in an asynchronous network. We then design an optimistic multiplication protocol. However, the challenge is that, now, no single party can be designated to secret-share the summands $[a]_p[b]_q$, since potentially corrupt parties may never secret-share the sum of the required summands. To get rid of this problem, we use an idea similar to what is used in our perfectly-secure protocol. That is, every party secret-shares the sum of all possible summands that it is capable of, while taking special care to ensure that no summand is shared more than once. This is followed by the parties probabilistically checking whether any cheating has occurred. In case any cheating is detected, the parties publicly identify and discard the corrupt party(ies). This is done by publicly reconstructing all the values involved. Upon discarding the corrupt party, the whole process is repeated again.

2 Preliminaries and Existing Asynchronous Primitives

We assume that the parties in $\mathcal{P} = \{P_1, \ldots, P_n\}$ are connected by pair-wise secure channels. The adversary Adv is assumed to be *malicious* and *static*, and decides the set of corrupt parties at the beginning of the protocol execution. Parties not under the control of Adv are called *honest*.

We assume that the parties want to compute a function f represented by a *publicly* known arithmetic circuit ckt over a finite field $\mathbb{F}$ consisting of linear and non-linear gates, with M being the number of multiplication gates. Without loss of generality, we assume that each $P_i \in \mathcal{P}$ has an input $x^{(i)}$ for f, and that all the parties want to learn the single output $y = f(x^{(1)}, \ldots, x^{(n)})$. We follow the asynchronous communication model of [4, 7]. The model does not put any restriction on the message delays and only guarantees that every message sent is delivered *eventually*. The sequence of message delivery is controlled by a scheduler, which is under the control of the adversary. Unlike the previous unconditionally-secure AMPC protocols [6, 3, 30, 13, 12], we prove the security of our protocols using the UC framework [8, 19, 9], based on the real-world/ideal-world paradigm. the details of which are presented next.

2.1 The Asynchronous Universal Composability (UC) Framework

The work of [15] has rigorously formalized the asynchronous network model with eventual message delivery in the UC framework, starting with the basic task of (asynchronous) point-to-point communication and formalizing asynchronous MPC. The same model was also followed in the work of [14], though the treatment was slightly *less* formal to avoid notational clutter and to make the protocols and proofs easier to understand. We also follow the same treatment of the UC model as [14] and recall the high-level description of the framework followed in [14]. Interested readers are referred to [26, 15] for the complete formal details.[3]

The work of [14] describes the asynchronous UC framework against threshold adversaries. We adapt the framework for the case of general adversaries. Informally, the security of a protocol is argued by "comparing" the capabilities of the adversary in two separate worlds. In the *real-world*, the parties exchange messages among themselves, computed as per a given protocol. In the *ideal-world*, the parties do not interact with *each other*, but with a *trusted* third-party (an ideal functionality), which enables the parties to obtain the result of the computation based on the inputs provided by the parties. Informally, a protocol is considered to be secure if whatever an adversary can do in the real protocol can be also done in the ideal-world.

The Asynchronous Real-World. An execution of a protocol Π in the real-world consists of n *interactive Turing machines* (ITMs) representing the parties in $\mathcal{P}$. Additionally, there is an ITM for representing the adversary Adv. Each ITM is initialized with its random coins and possible inputs. Additionally, Adv may have some auxiliary input z. Following the convention of [7], the protocol operates *asynchronously* by a sequence of *activations*, where at each point, a single ITM is active. Once activated, a party can perform some local computation, write on its output tape, or send messages to other parties. On the other hand, if the adversary is activated, it can send messages on the behalf of corrupt parties.[4]

To model the worst-case scenario, the adversary is given the provision to schedule the delivery of the messages exchanged between the parties. Once Adv delivers a message to some party, this party gets activated. The adversary cannot omit, change or inject messages. However, the adversary can reorder the messages sent by the honest parties. That is, it can decide which message will be delivered and when. Moreover, even though the adversary can delay the delivery of the messages *arbitrarily*, it *cannot* delay them *indefinitely*. That is, every message sent by a party is *eventually* delivered. These requirements on adversarial scheduling are formalized using the *eventual-delivery secure message-transmission* ideal functionality in [15].[5]

The protocol execution is complete once all honest parties obtain their respective outputs. We let $\mathsf{REAL}_{\Pi,\mathsf{Adv}(z),Z^\star}(\vec{x})$ denote the random variable consisting of the output of the honest parties and the view

[3]We stress that even though we prove the security of our protocols in the model proposed in [14], the proofs can be easily reworked to fit in the actual UC model proposed in [15].

[4]In [15], the order of activation is maintained and tracked in the protocols and proofs. However, doing so in the context of our protocols will make the proofs hard to read and understand and so we avoid doing that. However, we confirm that this will not violate the correctness of our UC claims and their proofs.

[5]In [15], while describing their protocols, the authors have used this functionality to capture the point-to-point communication between the parties. However, this brings in a lot of additional technicalities and notations. In the context of our protocols, sending every message through the asynchronous message-transmission functionality will make the protocols harder to read and understand. Hence, as done in [14], we will avoid communicating the messages in the protocol through the ideal asynchronous message-transmission functionality. However, we confirm that this will not violate the overall UC-security of our protocols.

of the adversary Adv during the execution of a protocol Π. Here, Adv controls parties in $Z^\star$ during the execution of protocol Π with inputs $\vec{x} = (x^{(1)}, \ldots, x^{(n)})$ for the parties (where party P_i has input $x^{(i)}$), and auxiliary input z for Adv.

The Asynchronous Ideal-World. A protocol in the ideal-world consists of n *dummy* parties $P_1, \ldots, P_n$, an ideal-world adversary $\mathcal{S}$ (also called *simulator*) and an ideal functionality $\mathcal{F}_{\mathsf{AMPC}}$. We consider *static* corruptions such that the set of corrupt parties $Z^\star$ is fixed at the beginning of the computation and is known to $\mathcal{S}$. The functionality $\mathcal{F}_{\mathsf{AMPC}}$ receives the inputs from the respective dummy parties, performs the desired computation f on the received inputs, and sends the outputs to the respective parties. The ideal-world adversary *does not* see and *cannot* delay the communication between the parties and $\mathcal{F}_{\mathsf{AMPC}}$. However, it can communicate with $\mathcal{F}_{\mathsf{AMPC}}$ on the behalf of corrupt parties.

Since $\mathcal{F}_{\mathsf{AMPC}}$ models the desired behaviour of a real-world protocol which is *asynchronous*, ideal functionalities must consider some inherent limitations to model the asynchronous communication model with eventual delivery. For example, in a real-world protocol, the adversary can decide when each honest party learns the output since it has full control over message scheduling. To model the notion of time in the ideal-world, [26] uses the concept of *number of activations*. Namely, once $\mathcal{F}_{\mathsf{AMPC}}$ has computed the output for some party, it *does not* ask "permission" from $\mathcal{S}$ to deliver it to the respective party. Instead, the corresponding party must "request" $\mathcal{F}_{\mathsf{AMPC}}$ for the output, which can be done only when the concerned party is active. Moreover, the adversary can "instruct" $\mathcal{F}_{\mathsf{AMPC}}$ to delay the output for each party by ignoring the corresponding requests, but only for a polynomial number of activations. If a party is activated sufficiently many times, the party will eventually receive the output from $\mathcal{F}_{\mathsf{AMPC}}$ and hence, ideal computation eventually completes. That is, each honest party eventually obtains its desired output. As in [14], we use the term "$\mathcal{F}_{\mathsf{AMPC}}$ *sends a request-based delayed output to* P_i", to describe the above interaction between the $\mathcal{F}_{\mathsf{AMPC}}, \mathcal{S}$ and P_i.

Another limitation is that in a real-world AMPC protocol, the (honest) parties *cannot* afford for all the parties to provide their input for the computation to avoid an endless wait, as the corrupt parties may decide not to provide their inputs. Hence, *every* AMPC protocol suffers from *input deprivation*, where inputs of a subset of potentially honest parties (which is decided by the choice of adversarial message scheduling) may get ignored during computation. Consequently, once a "core set" of parties $\mathcal{CS}$ provide their inputs for the computation, where $\mathcal{P} \setminus \mathcal{CS} \in \mathcal{Z}$, the parties have to start computing the function by assuming some default input for the left-over parties. To model this in the ideal-world, $\mathcal{S}$ is given the provision to decide the set $\mathcal{CS}$ of parties whose inputs should be taken into consideration by $\mathcal{F}_{\mathsf{AMPC}}$. We stress that $\mathcal{S}$ *cannot* delay sending $\mathcal{CS}$ to $\mathcal{F}_{\mathsf{AMPC}}$ indefinitely. This is because in the real-world protocol, Adv *cannot* prevent the honest parties from providing their inputs indefinitely. The formal description of $\mathcal{F}_{\mathsf{AMPC}}$ is available in Fig 1.

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

$\mathcal{F}_{\mathsf{AMPC}}$ proceeds as follows, running with the parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an adversary $\mathcal{S}$, and is parametrized by an n-party function $f : \mathbb{F}^n \rightarrow \mathbb{F}$ and an adversary structure $\mathcal{Z} \subset 2^{\mathcal{P}}$.

1. For each party $P_i \in \mathcal{P}$, initialize an input value $x^{(i)} = \perp$.
2. Upon receiving a message $(\mathsf{inp}, \mathsf{sid}, v)$ from some $P_i \in \mathcal{P}$ (or from $\mathcal{S}$ if P_i is *corrupt*), do the following:
   - Ignore the message if output has already been computed;
   - Else, set $x^{(i)} = v$ and send $(\mathsf{inp}, \mathsf{sid}, P_i)$ to $\mathcal{S}$.[a]
3. Upon receiving a message $(\mathsf{coreset}, \mathsf{sid}, \mathcal{CS})$ from $\mathcal{S}$, do the following:[b]
   - Ignore the message if $(\mathcal{P} \setminus \mathcal{CS}) \notin \mathcal{Z}$ or if output has already been computed;

- Else, record $\mathcal{CS}$ and set $x^{(i)} = 0$ for every $P_i \notin \mathcal{CS}$.[c]

4. If $\mathcal{CS}$ has been recorded and the value $x^{(i)}$ has been set to a value different from $\perp$ for every $P_i \in \mathcal{CS}$, then compute $y \stackrel{def}{=} f(x^{(1)}, \ldots, x^{(n)})$ and generate a request-based delayed output $(\mathsf{out}, \mathsf{sid}, (\mathcal{CS}, y))$ for every $P_i \in \mathcal{P}$.

[a] If P_i is corrupt, then no need to send $(\mathsf{inp}, \mathsf{sid}, P_i)$ to $\mathcal{S}$ as the input has been provided by $\mathcal{S}$ only.

[b] $\mathcal{S}$ cannot delay sending $\mathcal{CS}$ indefinitely; see the discussion before the description of the functionality.

[c] It is possible that for some $P_i \notin \mathcal{CS}$, the input has been set to a value different from 0 during step 1 and $x^{(i)}$ is now reset to 0. This models the scenario that in the real-world protocol, even if P_i is able to provide its input, P_i's inclusion to $\mathcal{CS}$ finally depends upon message scheduling, which is under adversarial control.

Figure 1: The ideal functionality for asynchronous secure multi-party computation for session id sid.

Similar to the real-world, we let $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{AMPC}}, \mathcal{S}(z), Z^\star}(\vec{x})$ denote the random variable consisting of the output of the honest parties and the view of the adversary $\mathcal{S}$, controlling the parties in $Z^\star$, with the parties having inputs $\vec{x} = (x^{(1)}, \ldots, x^{(n)})$ (where party P_i has input x_i), and auxiliary input z for $\mathcal{S}$.

We say that a real-world asynchronous protocol Π *securely realizes* $\mathcal{F}_{\mathsf{AMPC}}$ *with perfectly-security* if and only if for every real-world adversary Adv, there exists an ideal-world adversary $\mathcal{S}$ whose running time is polynomial in the running time of Adv, such that for every possible $Z^\star$, every possible $\vec{x} \in \mathbb{F}^n$ and every possible $z \in \{0,1\}^\star$, it holds that the random variables

$$\left\{\mathsf{REAL}_{\Pi, \mathsf{Adv}(z), Z^\star}(\vec{x})\right\} \quad \text{and} \quad \left\{\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{AMPC}}, \mathcal{S}(z), Z^\star}(\vec{x})\right\}$$

are identically distributed. That is, the random variables are perfectly-indistinguishable.

For statistically-secure AMPC, the parties and adversaries are parameterized with a statistical-security parameter κ, and the above random variables (which are viewed as ensembles, parameterized by κ) are required to be statistically-indistinguishable. That is, their statistical-distance should be a negligible function in κ.

The Universal-Composability (UC) Framework. While the real-world/ideal-world-based security paradigm is used to define the security of a protocol in the "stand-alone" setting, the more powerful UC framework [8, 9] is used to define the security of a protocol when multiple instances of the protocol might be running in parallel, possibly along with other protocols. Informally, the security in the UC-framework is still argued by comparing the real-world and the ideal-world. However, now, in both worlds, the computation takes place in the presence of an additional interactive process (modelled as an ITM) called the *environment* and denoted by Env. Roughly speaking, Env models the "external environment" in which protocol execution takes place. The interaction between Env and the various entities takes place as follows in the two worlds.

In the real-world, the environment gives inputs to the honest parties, receives their outputs, and can communicate with the adversary at any point during the execution. During the protocol execution, the environment gets activated first. Once activated, the environment can either activate one of the parties by providing some input or activate Adv by sending it a message. Once a party completes its operations upon getting activated, the control is returned to the environment. Once Adv gets activated, it can communicate with the environment (apart from sending messages to the honest parties). The environment also fully controls the corrupt parties that send all the messages they receive to Env, and follow the orders of Env. The protocol execution is completed once Env stops activating other parties, and outputs a single bit.

In the ideal-model, the environment Env gives inputs to the (dummy) honest parties, receives their outputs, and can communicate with $\mathcal{S}$ at any point during the execution. The dummy parties act as channels between Env and the functionality. That is, they send the inputs received from Env to functionality and transfer the output they receive from the functionality to Env. The activation sequence in this world is similar to the one in the real-world. The protocol execution is completed once Env stops activating other parties and outputs a single bit.

A protocol is said to be UC-secure with *perfect-security*, if for every real-world adversary Adv there exists a simulator $\mathcal{S}$, such that for any environment Env, the environment cannot distinguish the real-world from the ideal-world. On the other hand, the protocol is said to be UC-secure with *statistical-security*, if the environment cannot distinguish the real-world from the ideal-world, except with a probability which is a negligible function in the statistical-security parameter κ.

The Hybrid Model. In a $\mathcal{G}$-hybrid model, a protocol execution proceeds as in the real-world. However, the parties have access to an ideal functionality $\mathcal{G}$ for some specific task. During the protocol execution, the parties communicate with $\mathcal{G}$ as in the ideal-world. The UC framework guarantees that an ideal functionality in a hybrid model can be replaced with a protocol that UC-securely realizes $\mathcal{G}$. This is specifically due to the following composition theorem from [8, 9].

Theorem 2.1 ([8, 9])**.** *Let Π be a protocol that UC-securely realizes some functionality $\mathcal{F}$ in the $\mathcal{G}$-hybrid model and let ρ be a protocol that UC-securely realizes $\mathcal{G}$. Moreover, let Π^ρ denote the protocol that is obtained from Π by replacing every ideal call to $\mathcal{G}$ with the protocol ρ. Then Π^ρ UC-securely realizes $\mathcal{F}$ in the model where the parties do not have access to the functionality $\mathcal{G}$.*

2.2 Secret Sharing

In our protocols, we use a secret-sharing based on the one from [28], defined with respect to a *sharing specification* $\mathbb{S} = \{S_1, \ldots, S_h\} \stackrel{def}{=} \{\mathcal{P} \setminus Z | Z \in \mathcal{Z}\}$, where $h = |\mathbb{S}| = |\mathcal{Z}|$. This sharing specification $\mathbb{S}$ is $\mathcal{Z}$-*private*, meaning that for every $Z \in \mathcal{Z}$, there is an $S \in \mathbb{S}$ such that $Z \cap S = \emptyset$. Moreover, if $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition, then $\mathbb{S}$ satisfies the $\mathbb{Q}^{(3)}(\mathbb{S}, \mathcal{Z})$ condition. That is, for every $Z_{i_1}, Z_{i_2}, Z_{i_3} \in \mathcal{Z}$ and every $S \in \mathbb{S}$, the following holds:

$$S \not\subseteq Z_{i_1} \cup Z_{i_2} \cup Z_{i_3}.$$

In general, we say that $\mathbb{S}$ satisfies the $\mathbb{Q}^{(k)}(\mathbb{S}, \mathcal{Z})$ condition if for every $Z_{i_1}, \ldots, Z_{i_k} \in \mathcal{Z}$ and every $S \in \mathbb{S}$, the following holds:

$$S \not\subseteq Z_{i_1} \cup \ldots \cup Z_{i_k}.$$

Definition 2.2 ([28])**.** A value $s \in \mathbb{F}$ is said to be secret-shared with respect to $\mathbb{S}$, if there exist shares $s_1, \ldots, s_{|\mathbb{S}|} \in \mathbb{F}$, such that all the following hold.

– $s = s_1 + \ldots + s_{|\mathbb{S}|}$;
– For $q = 1, \ldots, |\mathbb{S}|$, the share s_q is known to every (honest) party in S_q.

The vector of shares corresponding to a secret-sharing of s is denoted by $[s]$, where $[s]_q$ denotes the q^{th} share. Note that a party P_i may hold multiple shares from $[s]$, depending upon the number of subsets in $\mathbb{S}$ in which it is present. Specifically, P_i will hold the shares $\{[s]_q\}_{P_i \in S_q}$. The above secret-sharing is *linear*, as $[c_1 s_1 + c_2 s_2] = c_1[s_1] + c_2[s_2]$ for any publicly-known $c_1, c_2 \in \mathbb{F}$. In general, the parties can *non-interactively* compute any linear function of secret-shared values, by applying the corresponding linear function on their respective shares of the function inputs.

Default Secret-Sharing. The perfectly-secure protocol Π_{PerDefSh} takes a *public* input $s \in \mathbb{F}$ and $\mathbb{S}$ to *non-interactively* generate $[s]$, where the parties collectively set $[s]_1 = s$ and $[s]_2 = \ldots = [s]_{|\mathbb{S}|} = 0$.

2.3 Existing Asynchronous Primitives

We next discuss the existing asynchronous primitives used in our protocols.

Asynchronous Reliable Broadcast (Acast). An Acast protocol allows a designated *sender* $P_S \in \mathcal{P}$ to identically send a message $m \in \{0,1\}^\ell$ to all the parties. If P_S is *honest*, then all honest parties eventually output m. On the other hand, if P_S is *corrupt* and some honest party outputs $m^\star$, then every other honest party eventually outputs $m^\star$. The above requirements are formalized by an ideal functionality $\mathcal{F}_{\mathsf{Acast}}$, presented in Fig 2. The functionality, upon receiving m from the sender P_S, performs a request-based delayed delivery of m to all the parties. Notice that if P_S is *corrupt*, then the functionality *may not* receive any message for delivery, in which case parties obtain no output. This models the fact that in any real-world Acast protocol, a potentially *corrupt* P_S *may not* invoke the protocol, in which case the (honest) parties obtain no output.

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

$\mathcal{F}_{\mathsf{Acast}}$ proceeds as follows, running with the parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an adversary $\mathcal{S}$, and is parametrized by an adversary structure $\mathcal{Z} \subset 2^{\mathcal{P}}$. Let $Z^\star$ denote the set of corrupt parties, where $Z^\star \in \mathcal{Z}$.

- Upon receiving $(\mathsf{sender}, \mathsf{Acast}, \mathsf{sid}, m)$ from $P_S \in \mathcal{P}$ (or from $\mathcal{S}$ if $P_S \in Z^\star$), do the following:
  - Send $(P_S, \mathsf{Acast}, \mathsf{sid}, m)$ to $\mathcal{S}$;[a]
  - Send a request-based delayed output $(P_S, \mathsf{Acast}, \mathsf{sid}, m)$ to each $P_i \in \mathcal{P} \setminus Z^\star$ (no need to send m to the parties in $Z^\star$, as $\mathcal{S}$ gets m on their behalf).

[a] If $P_S \in Z^\star$, then no need to send $(P_S, \mathsf{Acast}, \mathsf{sid}, m)$ to $\mathcal{S}$, as in this case m is received from $\mathcal{S}$ itself.

Figure 2: The ideal functionality for asynchronous reliable broadcast for session id sid.

In [27], a perfectly-secure Acast protocol Π_{Acast} is presented with a communication complexity of $\mathcal{O}(n^2\ell)$ bits, provided $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. The protocol is shown to be UC-secure in [11].

Remark 2.3. We note that the functionality $\mathcal{F}_{\mathsf{Acast}}$ is "one shot" per session id. That is, it allows the sender P_S to send at most one message per session id and a potentially *corrupt* P_S *cannot* send multiple messages for delivery to the functionality within the same session id. This is ensured by letting $\mathcal{F}_{\mathsf{Acast}}$ implicitly accepting at most one message from P_S for delivery for a given session id.

We also note that we implicitly assume that the identity of the designated sender P_S is known to all the participants. In our higher-level protocols where the parties will be calling $\mathcal{F}_{\mathsf{Acast}}$, the identity of P_S will be publicly known. To distinguish apart the messages exchanged during the various calls to $\mathcal{F}_{\mathsf{Acast}}$ with respect to different sender parties, one could tag all the messages exchanged with respect to a particular instance of $\mathcal{F}_{\mathsf{Acast}}$ with the identity of the corresponding sender party.

Asynchronous Byzantine Agreement (ABA). In a *synchronous* BA protocol [31], each party participates with an input bit to obtain an output bit. The protocol guarantees the following three properties.

- *Agreement*: The output bit of all honest parties is the same.
- *Validity*: If all honest parties have the same input bit, then this will be the common output bit.
- *Termination*: All honest parties eventually compute an output.

In an ABA protocol, the above requirements are slightly weakened, since all (honest) parties *may not* be able to provide their inputs to the protocol, as waiting for all the inputs may turn out to be an endless wait. Hence the decision is taken based on the inputs of a subset of parties $\mathcal{CS}$, where $\mathcal{P} \setminus \mathcal{CS} \in \mathcal{Z}$. Moreover, since the adversary can control the schedule of message delivery, it has full control in deciding the set $\mathcal{CS}$.

The formal specification of an ideal ABA functionality is presented in Fig 3, which is obtained by generalizing the corresponding ideal functionality against *threshold* adversaries, as presented in [15]. Intuitively, it can be considered as a special case of the ideal AMPC functionality (see Fig 1), which looks at the set of inputs provided by the set of parties in $\mathcal{CS}$, where $\mathcal{CS}$ is decided by the ideal-world adversary. If the input bits provided by all the honest parties in $\mathcal{CS}$ are the same, then it is set as the output bit. Else, the output bit is set to be the input bit provided by some corrupt party in $\mathcal{CS}$ (for example, the first corrupt party in $\mathcal{CS}$ according to lexicographic ordering). In the functionality, the input bits provided by various parties are considered to be the "votes" of the respective parties.

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

$\mathcal{F}_{\mathsf{ABA}}$ proceeds as follows, running with the parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an adversary $\mathcal{S}$, and is parametrized by an adversary-structure $\mathcal{Z} \subset 2^{\mathcal{P}}$. Let $Z^\star$ denote the set of corrupt parties, where $Z^\star \in \mathcal{Z}$ and let $\mathcal{H} = \mathcal{P} \setminus Z^\star$. For each party P_i, initialize an input value $x^{(i)} = \bot$.

1. Upon receiving a message $(\mathsf{vote}, \mathsf{sid}, b)$ from some $P_i \in \mathcal{P}$ (or from $\mathcal{S}$ if P_i is *corrupt*) where $b \in \{0, 1\}$, do the following:
   - Ignore the message if output has been already computed;
   - Else, set $x^{(i)} = b$ and send $(\mathsf{vote}, \mathsf{sid}, P_i, b)$ to $\mathcal{S}$.[a]
2. Upon receiving a message $(\mathsf{coreset}, \mathsf{sid}, \mathcal{CS}, b^\star)$ from $\mathcal{S}$ where $b^\star \in \{0, 1\}$, do the following:[b]
   - Ignore the message if $(\mathcal{P} \setminus \mathcal{CS}) \notin \mathcal{Z}$ or if output has been already computed;
   - Else, record $\mathcal{CS}$.
3. If the set $\mathcal{CS}$ has been recorded and the value $x^{(i)}$ has been set to a value different from $\bot$ for every $P_i \in \mathcal{CS}$, then compute the output y as follows and generate a request-based delayed output $(\mathsf{decide}, \mathsf{sid}, (\mathcal{CS}, y))$ for every $P_i \in \mathcal{P}$.
   - If $x^{(i)} = b$ holds for all $P_i \in (\mathcal{H} \cap \mathcal{CS})$, then set $y = b$.
   - Else if $\mathcal{CS} \cap Z^\star \neq \emptyset$, set $y = x^{(i)}$, where P_i is the party with the smallest index in $\mathcal{CS} \cap Z^\star$.
   – Else set $y = b^\star$.

[a] If $P_i \in Z^\star$, then no need to send $(\mathsf{vote}, \mathsf{sid}, P_i, b)$ to $\mathcal{S}$ as the input has been provided by $\mathcal{S}$ only.
[b] As in the case of the AMPC functionality $\mathcal{F}_{\mathsf{AMPC}}$, $\mathcal{S}$ cannot delay sending $\mathcal{CS}$ indefinitely.

Figure 3: The ideal functionality for asynchronous Byzantine agreement for session id sid.

From [17], every (deterministic) ABA protocol must have some non-terminating runs, where the parties may run the protocol forever, *without* obtaining any output. To circumvent this result, randomized ABA protocols are considered and the best we can hope for from such protocols is that the parties eventually obtain an output with probability 1, if they continue running the protocol (this property is called the *almost-surely termination* property). In [12], a perfectly-secure ABA protocol is presented, provided $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition, which holds for our perfectly-secure AMPC protocol. In the protocol, all honest parties eventually compute their output with probability 1 and the protocol incurs an expected communication of $\mathcal{O}(|\mathcal{Z}| \cdot (n^6 \log |\mathbb{F}| + n^8(\log n + \log |\mathcal{Z}|)))$ bits. We will use this ABA protocol for securely realizing $\mathcal{F}_{\mathsf{ABA}}$ in our perfectly-secure AMPC protocol.

Recently, [11] presented a perfectly-secure ABA protocol, provided $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition, which holds for our statistically-secure AMPC protocol. In the protocol, all honest parties eventually compute their output with probability 1 and the protocol incurs an expected communication of $\mathcal{O}(|\mathcal{Z}| \cdot (n^7 \log |\mathbb{F}| + n^8 \log n))$ bits. We will use this ABA protocol for securely realizing $\mathcal{F}_{\mathsf{ABA}}$ in our statistically-secure AMPC protocol.

We note that the security of the ABA protocols of [12, 11] are not proved in the UC model. This is because their main goal is to show the *feasibility* of ABA against generalized adversaries. However, we confirm that UC security proof of these protocols can be provided by bringing in additional technicalities, especially given the fact that we are considering static adversaries.

2.4 Some Asynchronous Ideal-World Functionalities

We end this section by discussing a few asynchronous ideal-world functionalities, which we later securely realize through various protocols.

Verifiable Secret-Sharing (VSS). A VSS protocol allows a designated *dealer* $P_{\mathsf{D}} \in \mathcal{P}$ to verifiably secret-share its input $s \in \mathbb{F}$. If P_{D} is *honest*, then the honest parties eventually output $[s]$. The verifiability property guarantees that if P_{D} is *corrupt* and some honest party completes the protocol, then all honest parties eventually complete the protocol with a secret-sharing of some value. These requirements are formalized through the functionality $\mathcal{F}_{\mathsf{VSS}}$ (Fig 4). The functionality, upon receiving a vector of shares from P_{D}, distributes the appropriate shares to the respective parties. The dealer's input is defined *implicitly* as the sum of provided shares. Looking ahead, we will use $\mathcal{F}_{\mathsf{VSS}}$ in our protocols as follows: P_{D} on having the input s, sends a random vector of shares $(s_1, \ldots, s_{|\mathbb{S}|})$ to $\mathcal{F}_{\mathsf{VSS}}$ where $s_1 + \ldots + s_{|\mathbb{S}|} = s$. If P_{D} is *honest*, then the probability distribution of the shares learnt by Adv will be *independent* of the dealer's input, since $\mathbb{S}$ is *$\mathcal{Z}$-private*. Note that if P_{D} is *corrupt*, then it may not provide any vector of shares to the functionality, in which case the honest parties obtain no output. This models the fact that in the asynchronous setting, the honest parties need not have any output in the real-world VSS protocol, if a potentially *corrupt* P_{D} *does not* invoke the protocol in the first place.

We note that in the functionality $\mathcal{F}_{\mathsf{VSS}}$, we let the dealer P_{D} select the shares, corresponding to its input s, instead of the functionality picking the shares. One could instead consider a variant of $\mathcal{F}_{\mathsf{VSS}}$, where if P_{D} is *honest*, then it provides its input s to the functionality, who then picks random shares corresponding to s and distributes them to the respective parties. And if P_{D} is *corrupt*, then the functionality receives the full vector of shares from P_{D} and distributes them.[6] Looking ahead, our VSS protocol can be easily modified to securely realize even this variant of $\mathcal{F}_{\mathsf{VSS}}$. The reason we let P_{D} provide the full vector of shares is that it "unifies" the case of both honest as well as a corrupt P_{D}, since $\mathcal{F}_{\mathsf{VSS}}$ will be performing the same set of actions, *irrespective* of P_{D}. As mentioned earlier, the way we will invoke the functionality $\mathcal{F}_{\mathsf{VSS}}$ in our higher-level protocols, it will be guaranteed that if P_{D} is *honest*, then its input s will be *randomly* secret-shared, since P_{D} will be providing a random vector of shares for distribution to $\mathcal{F}_{\mathsf{VSS}}$. So it makes no difference whether an *honest* P_{D} provides a random vector of shares for its secret or whether the functionality $\mathcal{F}_{\mathsf{VSS}}$ picks random shares on the behalf of an *honest* P_{D}. In either case, the full vector of shares will be randomly distributed, subject to the condition that they sum up to s.

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

$\mathcal{F}_{\mathsf{VSS}}$ proceeds as follows for each party $P_i \in \mathcal{P}$ and an adversary $\mathcal{S}$, and is parametrized by the adversary structure $\mathcal{Z}$, sharing specification $\mathbb{S} = \{S_1, \ldots S_h\} = \{\mathcal{P} \setminus Z | Z \in \mathcal{Z}\}$ and a dealer $P_{\mathsf{D}} \in \mathbb{S}$. Let $Z^\star \in \mathcal{Z}$ be the set of corrupt parties.

- On receiving $(\mathsf{dealer}, \mathsf{sid}, P_{\mathsf{D}}, (s_1, \ldots, s_h))$ from P_{D} (or from $\mathcal{S}$ if $P_{\mathsf{D}} \in Z^\star$), do the following.
  - Set $s = \sum_{q=1,\ldots,h} s_q$.
  - For $q = 1, \ldots, h$, set $[s]_q = s_q$.
  - If $P_{\mathsf{D}} \in Z^\star$, then generate a request-based delayed output $(\mathsf{share}, \mathsf{sid}, P_{\mathsf{D}}, \{[s]_q\}_{P_i \in S_q})$ for each $P_i \notin Z^\star$. [a]
  - Else generate a request-based delayed output $(\mathsf{share}, \mathsf{sid}, P_{\mathsf{D}}, \{[s]_q\}_{P_i \in S_q})$ for each $P_i \in \mathcal{P}$.

[a] If P_{D} is *corrupt*, then $\mathcal{S}$ may not send any input to $\mathcal{F}_{\mathsf{VSS}}$, in which case the functionality will not generate any output.

Figure 4: The ideal functionality for VSS for session id sid.

[6] In the latter case, we *cannot* let the functionality pick random shares on behalf of P_{D}'s input. This is because in the real-world VSS protocol, a *corrupt* P_{D} *may not* select random shares for secret-sharing its input.

Remark 2.4. We note that the functionality $\mathcal{F}_{\mathsf{VSS}}$ is "one shot" per session id. That is, it allows the dealer to send at most one set of shares per session id. A potentially *corrupt* dealer cannot send multiple sets of shares to the functionality within the same session id. This is ensured by $\mathcal{F}_{\mathsf{VSS}}$ implicitly accepting at most one set of shares from the dealer for processing for a given session id.

We also note that we implicitly assume that the identity of the designated dealer P_{D} is known to all the participants. In our higher-level protocols where the parties will be calling $\mathcal{F}_{\mathsf{VSS}}$, the identity of P_{D} will be publicly known. To distinguish apart the messages exchanged during the various calls to $\mathcal{F}_{\mathsf{VSS}}$ with respect to different dealers, one could tag all the messages exchanged with respect to a particular instance of $\mathcal{F}_{\mathsf{VSS}}$ with the identity of the corresponding dealer.

Triple-Generation Functionality. The ideal functionality $\mathcal{F}_{\mathsf{Triples}}$ (Fig 5) models the pre-processing phase of our AMPC protocols. The functionality generates secret-sharing of M random multiplication-triples over $\mathbb{F}$, which are random from the point of view of the adversary.[7] The functionality allows the ideal-world adversary to specify the shares for each of the output triples on the behalf of corrupt parties. The functionality then "completes" the sharing of all the triples randomly, while keeping them "consistent" with the shares specified by the adversary.[8]

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

$\mathcal{F}_{\mathsf{Triples}}$ proceeds as follows, running with the parties $\mathcal{P}$ and an adversary $\mathcal{S}$, and is parametrized by an adversary-structure $\mathcal{Z}$, a $\mathcal{Z}$-*private* sharing specification $\mathbb{S} = \{S_1, \ldots, S_h\} = \{\mathcal{P} \setminus Z | Z \in \mathcal{Z}\}$ and the number of multiplication gates M in ckt. Let $Z^\star \in \mathcal{Z}$ denote the set of corrupt parties.

- If there is a set of parties $\mathcal{A}$ such that $\mathcal{P} \setminus \mathcal{A} \in \mathcal{Z}$ and every $P_i \in \mathcal{A}$ has sent the message $(\mathsf{triples}, \mathsf{sid}, P_i)$, then send $(\mathsf{triples}, \mathsf{sid}, \mathcal{A})$ to $\mathcal{S}$ and prepare the output as follows.
  - Generate secret-sharing of M random multiplication-triples. To generate one such sharing, randomly select $a, b \in \mathbb{F}$, compute $c = ab$ and execute the steps labelled **Single Sharing Generation** for a, b and c.
  - Let $\{([a^{(\ell)}], [b^{(\ell)}], [c^{(\ell)}])\}_{\ell \in \{1,\ldots,M\}}$ be the resultant secret-sharing of the multiplication-triples. Send a request-based delayed output $(\mathsf{tripleshares}, \mathsf{sid}, \{[a^{(\ell)}]_q, [b^{(\ell)}]_q, [c^{(\ell)}]_q\}_{\ell \in \{1,\ldots,M\}, P_i \in S_q})$ to each $P_i \in \mathcal{P} \setminus Z^\star$ (no need to send the respective shares to the parties in $Z^\star$, as $\mathcal{S}$ already has the shares of all the corrupt parties).

  Single Sharing Generation: Do the following to generate a secret-sharing of a given value s.
  - Upon receiving $(\mathsf{shares}, \mathsf{sid}, \{s_q\}_{S_q \cap Z^\star \neq \emptyset})$ from $\mathcal{S}$, randomly select $s_q \in \mathbb{F}$ corresponding to each $S_q \in \mathbb{S}$ for which $S_q \cap Z^\star = \emptyset$, such that $\sum\limits_{S_q \cap Z^\star \neq \emptyset} s_q + \sum\limits_{S_q \cap Z^\star = \emptyset} s_q = s$ holds. [a] For $q = 1, \ldots, h$, set $[s]_q = s_q$.

[a] $\mathcal{S}$ *cannot* delay sending the shares on the behalf of the corrupt parties indefinitely as, in our real-world protocol, the adversary *cannot* indefinitely delay the generation of secret-shared multiplication-triples.

Figure 5: Ideal functionality for asynchronous pre-processing phase with session id sid.

3 Perfectly-Secure Pre-Processing Phase Protocol with $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ Condition

In this section, we present a perfectly-secure protocol for securely realizing the functionality $\mathcal{F}_{\mathsf{Triples}}$. For designing the protocol, we need two building blocks: a perfectly-secure VSS protocol and a perfectly-secure multiplication protocol, which we discuss next.

[7] Recall that M is the number of multiplication gates in the circuit ckt.

[8] This provision is made because in our pre-processing phase protocol, the real-world adversary will have full control over the shares of the corrupt parties corresponding to the random multiplication-triples generated in the protocol.

3.1 Perfectly-Secure Verifiable Secret Sharing (VSS)

In [12], a perfectly-secure VSS protocol was presented. We recall and present the protocol here for two reasons. Firstly, we will show how to slightly modify the protocol to improve its communication complexity. Secondly, and more importantly, in [12], the UC-security of the protocol was not proved. Since we are aiming to prove the UC-security of our AMPC protocol where the VSS protocol will be used, we give a proof of the UC-security of the VSS protocol.

In the perfectly-secure VSS protocol Π_{PVSS}, the input for the dealer P_{D} is a vector of shares $(s_1, \ldots, s_h)$, the goal being to ensure that the parties output a secret-sharing of $s \stackrel{def}{=} s_1 + \ldots + s_h$, such that $[s]_q = s_q$, for each $S_q \in \mathbb{S}$. The protocol guarantees that even if P_{D} is *corrupt*, if some honest party completes the protocol, then every honest party eventually completes the protocol such that there exists some value which has been secret-shared by P_{D}.

The high-level idea of the protocol is as follows: the dealer gives the share s_q to all the parties in the set $S_q \in \mathbb{S}$. To verify whether the dealer has distributed the *same* share to all the parties in S_q, the parties in S_q perform pairwise consistency tests of the supposedly common share and *publicly* announce the result. Next, the parties check if there exists a subset of "core" parties $\mathcal{C}_q$, where $S_q \setminus \mathcal{C}_q \in \mathcal{Z}$, who have confirmed the pairwise consistency of their supposedly common share. Such a subset $\mathcal{C}_q$ is guaranteed for an *honest* dealer, as the set of honest parties in S_q always constitutes a candidate set for $\mathcal{C}_q$. To ensure that all honest parties have the same version of the core sets $\mathcal{C}_1, \ldots, \mathcal{C}_h$, the dealer is assigned the task of identifying these sets based on the results of the pairwise consistency tests, and making them public. Once the core sets are identified and verified, it is guaranteed that the dealer has distributed some common share to all *honest* parties within $\mathcal{C}_q$. The next goal is to ensure that even the honest parties in $S_q \setminus \mathcal{C}_q$ get this common share, which is required as per the semantics of our secret-sharing. For this, the (honest) parties in $S_q \setminus \mathcal{C}_q$ "filter" out the supposedly common shares received during the pairwise consistency tests and ensure that they obtain the common share held by the honest parties in $\mathcal{C}_q$. Protocol Π_{PVSS} is presented in Fig 6.

Protocol $\Pi_{\mathsf{PVSS}}(\mathbb{S})$

- **Distribution of Shares by** P_{D} : If P_i is the dealer P_{D}, then execute the following steps.
  1. On having the shares $s_1, \ldots, s_h \in \mathbb{F}$, send $(\mathsf{dist}, \mathsf{sid}, P_{\mathsf{D}}, q, [s]_q)$ to all the parties $P_i \in S_q$, corresponding to each $S_q \in \mathbb{S}$, where $s \stackrel{def}{=} s_1 + \ldots + s_h$ and $[s]_q = s_q$.
- **Pairwise Consistency Tests and Public Announcement of Results** : For each $S_q \in \mathbb{S}$, if $P_i \in S_q$, then execute the following steps.
  1. Upon receiving $(\mathsf{dist}, \mathsf{sid}, P_{\mathsf{D}}, q, s_{qi})$ from D, send $(\mathsf{test}, \mathsf{sid}, P_{\mathsf{D}}, q, s_{qi})$ to every party $P_j \in S_q$.
  2. Upon receiving $(\mathsf{test}, \mathsf{sid}, P_{\mathsf{D}}, q, s_{qj})$ from $P_j \in S_q$, send $(\mathsf{sender}, \mathsf{Acast}, \mathsf{sid}_{ij}^{(P_{\mathsf{D}},q)}, \mathsf{OK}_q(i,j))$ to $\mathcal{F}_{\mathsf{Acast}}$ if $s_{qi} = s_{qj}$, where $\mathsf{sid}_{ij}^{(P_{\mathsf{D}},q)} = \mathsf{sid}||P_{\mathsf{D}}||q||i||j$.[a]
- **Constructing Consistency Graph** : For each $S_q \in \mathbb{S}$, execute the following steps.
  1. Initialize $\mathcal{C}_q$ to $\emptyset$. Construct an undirected consistency graph $G_q^{(i)}$ with S_q as the vertex set.
  2. For every ordered pair of parties (P_j, P_k) where $P_j, P_k \in S_q$, keep requesting for an output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{jk}^{(P_{\mathsf{D}},q)}$, till an output is received.
  3. Add the edge (P_j, P_k) to $G_q^{(i)}$ if outputs $(P_j, \mathsf{Acast}, \mathsf{sid}_{jk}^{(P_{\mathsf{D}},q)}, \mathsf{OK}_q(j,k))$ and $(P_k, \mathsf{Acast}, \mathsf{sid}_{kj}^{(P_{\mathsf{D}},q)}, \mathsf{OK}_q(k,j))$ are received from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{jk}^{(P_{\mathsf{D}},q)}$ and $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{kj}^{(P_{\mathsf{D}},q)}$ respectively.
- **Identification of Core Sets and Public Announcements** : If P_i is the dealer P_{D}, then execute the following steps.
  1. For each $S_q \in \mathbb{S}$, check if there exists a subset of parties $\mathcal{W}_q \subseteq S_q$, such that $S_q \setminus \mathcal{W}_q \in \mathcal{Z}$ and the parties in $\mathcal{W}_q$ form a clique in the consistency graph G_q^{D}. If such a $\mathcal{W}_q$ exists, then assign $\mathcal{C}_q := \mathcal{W}_q$.

2. Upon computing non-empty sets $\mathcal{C}_1, \ldots, \mathcal{C}_h$, send $(\mathsf{sender}, \mathsf{Acast}, \mathsf{sid}_{P_\mathsf{D}}, \{\mathcal{C}_q\}_{S_q \in \mathbb{S}})$ to $\mathcal{F}_{\mathsf{Acast}}$, where $\mathsf{sid}_{P_\mathsf{D}} = \mathsf{sid}||P_\mathsf{D}$.

- **Share Computation** : Execute the following steps.

1. For each $S_q \in \mathbb{S}$ such that $P_i \in S_q$, initialize $[s]_q$ to $\perp$.
2. Keep requesting for an output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{P_\mathsf{D}}$ until an output is received.
3. Upon receiving an output $(\mathsf{sender}, \mathsf{Acast}, \mathsf{sid}_{P_\mathsf{D}}, \{\mathcal{C}_q\}_{S_q \in \mathbb{S}})$ from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{P_\mathsf{D}}$, wait until the parties in $\mathcal{C}_q$ form a clique in $G_q^{(i)}$, for $q = 1, \ldots, h$. Then, verify if $S_q \setminus \mathcal{C}_q \in \mathcal{Z}$, for each $q = 1, \ldots, h$. If the verification is successful, then proceed to compute the output as follows.
   i. Corresponding to each $\mathcal{C}_q$ such that $P_i \in \mathcal{C}_q$, set $[s]_q := s_{qi}$.
   ii. Corresponding to each $\mathcal{C}_q$ such that $P_i \notin \mathcal{C}_q$, set $[s]_q := s_q$, where $(\mathsf{test}, \mathsf{sid}, P_\mathsf{D}, q, s_q)$ is received from a set of parties $\mathcal{C}'_q$ such that $\mathcal{C}_q \setminus \mathcal{C}'_q \in \mathcal{Z}$.
4. Once $[s]_q \neq \perp$ for each $S_q \in \mathbb{S}$ such that $P_i \in S_q$, output $(\mathsf{share}, \mathsf{sid}, P_\mathsf{D}, \{[s]_q\}_{P_i \in S_q})$.

[a] The notation $\mathsf{sid}_{ij}^{(P_\mathsf{D}, q)}$ is used here to distinguish among the different calls to $\mathcal{F}_{\mathsf{Acast}}$ within the session sid.

Figure 6: The perfectly-secure VSS protocol for realizing $\mathcal{F}_{\mathsf{VSS}}$ in the $\mathcal{F}_{\mathsf{Acast}}$-hybrid model. The above steps are executed by every $P_i \in \mathcal{P}$.

Remark 3.1. We stress that if P_D is *corrupt*, then the honest parties *may not* get any output in the protocol Π_{PVSS}, if P_D *does not* make public valid core sets. This is fine as per the semantics of the functionality $\mathcal{F}_{\mathsf{VSS}}$, since the functionality $\mathcal{F}_{\mathsf{VSS}}$ is not "obliged" to distribute any shares to the honest parties if P_D is *corrupt*. On the other hand, if P_D is *honest*, then it will eventually compute and broadcast valid core sets, since the set of honest parties will eventually satisfy all the required conditions of valid core sets.

We next prove the security of the protocol Π_{PVSS}.

Theorem 3.2. *Consider a static malicious adversary* Adv *characterized by an adversary-structure* $\mathcal{Z}$*, satisfying the* $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ *condition and let* $\mathbb{S} = \{S_1, \ldots, S_h\} \stackrel{def}{=} \{\mathcal{P} \setminus Z | Z \in \mathcal{Z}\}$ *be the sharing specification. Then protocol* Π_{PVSS} *UC-securely realizes the functionality* $\mathcal{F}_{\mathsf{VSS}}$ *with perfect security in the* $\mathcal{F}_{\mathsf{Acast}}$*-hybrid model, in the presence of* Adv.

Proof. Let Adv be an arbitrary adversary corrupting a set of parties $Z^\star \in \mathcal{Z}$. Let Env be an arbitrary environment. We show the existence of a simulator $\mathcal{S}_{\mathsf{PVSS}}$, such that for any $Z^\star \in \mathcal{Z}$, the outputs of the honest parties and the view of the adversary in the protocol Π_{PVSS} is indistinguishable from the outputs of the honest parties and the view of the adversary in an execution in the ideal world involving $\mathcal{S}_{\mathsf{PVSS}}$ and $\mathcal{F}_{\mathsf{VSS}}$. The steps of the simulator will be different depending on whether the dealer is corrupt of honest.

If the dealer is *honest*, then the simulator interacts with $\mathcal{F}_{\mathsf{VSS}}$ and receives the shares of the corrupt parties corresponding to the sets $S_q \in \mathbb{S}$ which they are part of. With these shares, the simulator then plays the role of the dealer as well as the honest parties, as per the steps of Π_{PVSS}, and interacts with Adv. The simulator also plays the role of $\mathcal{F}_{\mathsf{Acast}}$. If Adv queries $\mathcal{F}_{\mathsf{Acast}}$ for the result of any pairwise consistency test involving an honest party, the simulator provides the appropriate result. In addition, the simulator records the result of any test involving corrupt parties which Adv sends to $\mathcal{F}_{\mathsf{Acast}}$. Based on the results of these pairwise consistency tests, the simulator finds the core sets for each S_q and sends these to Adv upon request.

If the dealer is *corrupt*, the simulator plays the role of honest parties and interacts with Adv, as per the steps of Π_{PVSS}. This involves recording shares which Adv distributes to any honest party (on the behalf of the dealer), as well as performing pairwise consistency tests on their behalf. If Adv sends core sets for each $S_q \in \mathbb{S}$ as input to $\mathcal{F}_{\mathsf{Acast}}$, then the simulator checks if these are *valid*, and accordingly, sends the shares held by *honest* parties in these core sets as the input shares to $\mathcal{F}_{\mathsf{VSS}}$ on the behalf of the dealer. The simulator is presented in Figure 7.

Simulator $\mathcal{S}_{\mathsf{PVSS}}$

$\mathcal{S}_{\mathsf{PVSS}}$ constructs virtual real-world honest parties and invokes the real-world adversary Adv. The simulator simulates the view of Adv, namely its communication with Env, the messages sent by the honest parties and the interaction with $\mathcal{F}_{\mathsf{Acast}}$. In order to simulate Env, the simulator $\mathcal{S}_{\mathsf{PVSS}}$ forwards every message it receives from Env to Adv and vice-versa. The simulator then simulates the various phases of the protocol as follows, depending upon whether the dealer is honest or corrupt.

Simulation When P_{D} is Honest

Interaction with $\mathcal{F}_{\mathsf{VSS}}$: The simulator interacts with the functionality $\mathcal{F}_{\mathsf{VSS}}$ and receives a request based delayed output $(\mathsf{share}, \mathsf{sid}, P_{\mathsf{D}}, \{[s]_q\}_{S_q \cap Z^\star \neq \emptyset})$, on the behalf of the parties in $Z^\star$.

Distribution of Shares by P_{D}: On the behalf of the dealer, the simulator sends $(\mathsf{dist}, \mathsf{sid}, P_{\mathsf{D}}, q, [s]_q)$ to Adv, corresponding to every $P_i \in Z^\star \cap S_q$.

Pairwise Consistency Tests: For each $S_q \in \mathbb{S}$ such that $S_q \cap Z^\star \neq \emptyset$, corresponding to each $P_i \in S_q \cap Z^\star$, the simulator does the following.

- On the behalf of every party $P_j \in S_q \setminus Z^\star$, send $(\mathsf{test}, \mathsf{sid}, P_{\mathsf{D}}, q, s_{qj})$ to Adv, where $s_{qj} = [s]_q$.
- If Adv sends $(\mathsf{test}, \mathsf{sid}, P_{\mathsf{D}}, q, s_{qi})$ on the behalf of P_i to any $P_j \in S_q$, then record it.

Announcing Results of Consistency Tests:

- If for any $S_q \in \mathbb{S}$, Adv requests an output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{ij}^{(P_{\mathsf{D}},q)}$ corresponding to parties $P_i \in S_q \setminus Z^\star$ and $P_j \in S_q$, then the simulator provides output on the behalf of $\mathcal{F}_{\mathsf{Acast}}$ as follows.
  - If $P_j \in S_q \setminus Z^\star$, then send the output $(P_i, \mathsf{Acast}, \mathsf{sid}_{ij}^{(P_{\mathsf{D}},q)}, \mathsf{OK}_q(i,j))$.
  - If $P_j \in (S_q \cap Z^\star)$, then send the output $(P_i, \mathsf{Acast}, \mathsf{sid}_{ij}^{(P_{\mathsf{D}},q)}, \mathsf{OK}_q(i,j))$, if the message $(\mathsf{test}, \mathsf{sid}, P_{\mathsf{D}}, q, s_{qj})$ has been recorded on the behalf of P_j for party P_i and $s_{qj} = [s]_q$ holds.
- If for any $S_q \in \mathbb{S}$ and any $P_i \in S_q \cap Z^\star$, Adv sends $(P_i, \mathsf{Acast}, \mathsf{sid}_{ij}^{(P_{\mathsf{D}},q)}, \mathsf{OK}_q(i,j))$ to $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{ij}^{(P_{\mathsf{D}},q)}$ on the behalf of P_i for any $P_j \in S_q$, then the simulator records it. Moreover, if Adv requests for an output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{ij}^{(P_{\mathsf{D}},q)}$, then the simulator sends the output $(P_i, \mathsf{Acast}, \mathsf{sid}_{ij}^{(P_{\mathsf{D}},q)}, \mathsf{OK}_q(i,j))$ on the behalf of $\mathcal{F}_{\mathsf{Acast}}$.

Construction of Core Sets and Public Announcement:

- For each $S_q \in \mathbb{S}$, the simulator plays the role of P_{D} and adds the edge (P_i, P_j) to the graph G_q^{D} over the vertex set S_q, if the following hold.
  - $P_i, P_j \in S_q$.
  - One of the following is true.
    - $P_i, P_j \in S_q \setminus Z^\star$.
    - If $P_i \in S_q \cap Z^\star$ and $P_j \in S_q \setminus Z^\star$, then the simulator has recorded $(P_i, \mathsf{Acast}, \mathsf{sid}_{ij}^{(P_{\mathsf{D}},q)}, \mathsf{OK}_q(i,j))$ sent by Adv on the behalf of P_i to $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{ij}^{(P_{\mathsf{D}},q)}$, and recorded $(\mathsf{test}, \mathsf{sid}, P_{\mathsf{D}}, q, s_{qi})$ on the behalf of P_i for P_j such that $s_{qi} = [s]_q$.
    - If $P_i, P_j \in S_q \cap Z^\star$, then the simulator has recorded $(P_i, \mathsf{Acast}, \mathsf{sid}_{ij}^{(q)}, \mathsf{OK}_q(i,j))$ and $(P_j, \mathsf{Acast}, \mathsf{sid}_{ji}^{(q)}, \mathsf{OK}_q(j,i))$ sent by Adv on behalf P_i and P_j to $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{ij}^{(P_{\mathsf{D}},q)}$ and $\mathsf{sid}_{ji}^{(P_{\mathsf{D}},q)}$ respectively.
- For each $S_q \in \mathbb{S}$, the simulator finds the set $\mathcal{C}_q$ which forms a clique in G_q^{D}, such that $S_q \setminus \mathcal{C}_q \in \mathcal{Z}$. When Adv requests output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{P_{\mathsf{D}}}$, the simulator sends the output $(\mathsf{sender}, \mathsf{Acast}, \mathsf{sid}_{P_{\mathsf{D}}}, \{\mathcal{C}_q\}_{S_q \in \mathbb{S}})$ on the behalf of $\mathcal{F}_{\mathsf{Acast}}$.

Simulation When P_{D} is Corrupt

In this case, the simulator $\mathcal{S}_{\mathsf{PVSS}}$ interacts with Adv during the various phases of Π_{PVSS} as follows.

Distribution of shares by P_{D}: For $q = 1, \ldots, h$, if Adv sends $(\mathsf{dist}, \mathsf{sid}, P_{\mathsf{D}}, q, v)$ on the behalf of P_{D} to any party $P_i \in S_q \setminus Z^\star$, then the simulator records it and sets s_{qi} to be v.

Pairwise Consistency Tests: For each $S_q \in \mathbb{S}$ such that $S_q \cap Z^\star \neq \emptyset$, corresponding to each party $P_i \in S_q \cap Z^\star$ and each $P_j \in S_q \setminus Z^\star$, the simulator does the following.

- If s_{qj} has been set to some value, then send $(\mathsf{test}, \mathsf{sid}, P_{\mathsf{D}}, q, s_{qj})$ to Adv on the behalf of P_j.
- If Adv sends $(\mathsf{test}, \mathsf{sid}, P_{\mathsf{D}}, q, s_{qi})$ on the behalf of P_i to P_j, then record it.

Announcing Results of Consistency Tests:

- If for any $S_q \in \mathbb{S}$, Adv requests an output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{ij}^{(P_{\mathsf{D}},q)}$ corresponding to parties $P_i \in S_q \setminus Z^\star$ and $P_j \in S_q$, then the simulator provides the output on the behalf of $\mathcal{F}_{\mathsf{Acast}}$ as follows, if s_{qi} has been set to some value.
  - If $P_j \in S_q \setminus Z^\star$, then send the output $(P_i, \mathsf{Acast}, \mathsf{sid}_{ij}^{(P_{\mathsf{D}},q)}, \mathsf{OK}_q(i,j))$, if s_{qj} has been set to some value and $s_{qi} = s_{qj}$ holds.
  - If $P_j \in S_q \cap Z^\star$, then send the output $(P_i, \mathsf{Acast}, \mathsf{sid}_{ij}^{(P_{\mathsf{D}},q)}, \mathsf{OK}_q(i,j))$, if $(\mathsf{test}, \mathsf{sid}, P_{\mathsf{D}}, q, s_{qj})$ sent by Adv on the behalf of P_j to P_i has been recorded and $s_{qj} = s_{qi}$ holds.
- If for any $S_q \in \mathbb{S}$ and any $P_i \in S_q \cap Z^\star$, Adv sends $(P_i, \mathsf{Acast}, \mathsf{sid}_{ij}^{(P_{\mathsf{D}},q)}, \mathsf{OK}_q(i,j))$ to $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{ij}^{(P_{\mathsf{D}},q)}$ on the behalf of P_i for any $P_j \in S_q$, then the simulator records it. Moreover, if Adv requests for an output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{ij}^{(P_{\mathsf{D}},q)}$, then the simulator sends the output $(P_i, \mathsf{Acast}, \mathsf{sid}_{ij}^{(P_{\mathsf{D}},q)}, \mathsf{OK}_q(i,j))$ on the behalf of $\mathcal{F}_{\mathsf{Acast}}$.

Construction of Core Sets: For each $S_q \in \mathbb{S}$, the simulator plays the role of the honest parties $P_i \in S_q \setminus Z^\star$ and adds the edge (P_j, P_k) to the graph $G_q^{(i)}$ over vertex set S_q, if the following hold.

- $P_j, P_k \in S_q$.
- One of the following is true.
  - If $P_j, P_k \in S_q \setminus Z^\star$, then the simulator has set s_{qj} and s_{qk} to some values, such that $s_{qj} = s_{qk}$.
  - If $P_j \in S_q \cap Z^\star$ and $P_k \in S_q \setminus Z^\star$, then the simulator has recorded $(P_j, \mathsf{Acast}, \mathsf{sid}_{jk}^{(P_{\mathsf{D}},q)}, \mathsf{OK}_q(j,k))$ sent by Adv on the behalf of P_j to $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{jk}^{(P_{\mathsf{D}},q)}$, and recorded $(\mathsf{test}, \mathsf{sid}, P_{\mathsf{D}}, q, s_{qj})$ on the behalf of P_j for P_k and has set s_{qk} to a value such that $s_{qj} = s_{qk}$.
  - If $P_j, P_k \in S_q \cap Z^\star$, then the simulator has recorded $(P_j, \mathsf{Acast}, \mathsf{sid}_{jk}^{(P_{\mathsf{D}},q)}, \mathsf{OK}_q(j,k))$ and $(P_k, \mathsf{Acast}, \mathsf{sid}_{kj}^{(P_{\mathsf{D}},q)}, \mathsf{OK}_q(k,j))$ sent by Adv on behalf of P_j and P_k respectively to $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{jk}^{(P_{\mathsf{D}},q)}$ and $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{kj}^{(P_{\mathsf{D}},q)}$.

Verification of Core Sets and Interaction with $\mathcal{F}_{\mathsf{VSS}}$:

- If Adv sends $(\mathsf{sender}, \mathsf{Acast}, \mathsf{sid}_{P_{\mathsf{D}}}, \{\mathcal{C}_q\}_{S_q \in \mathbb{S}})$ to $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{P_{\mathsf{D}}}$ on the behalf of P_{D}, then the simulator records it. Moreover, if Adv requests an output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{P_{\mathsf{D}}}$, then on the behalf of $\mathcal{F}_{\mathsf{Acast}}$, the simulator sends the output $(P_{\mathsf{D}}, \mathsf{Acast}, \mathsf{sid}_{P_{\mathsf{D}}}, \{\mathcal{C}_q\}_{S_q \in \mathbb{S}})$.
- If simulator has recorded the sets $\{\mathcal{C}_q\}_{S_q \in \mathbb{S}}$, then it plays the role of the honest parties and verifies if $\mathcal{C}_1, \ldots, \mathcal{C}_h$ are valid by checking if each $S_q \setminus \mathcal{C}_q \in \mathcal{Z}$ and if each $\mathcal{C}_q$ constitutes a clique in the graph $G_q^{(i)}$ of every party $P_i \in \mathcal{P} \setminus Z^\star$. If $\mathcal{C}_1, \ldots, \mathcal{C}_h$ are valid, then the simulator sends $(\mathsf{share}, \mathsf{sid}, P_{\mathsf{D}}, \{s_q\}_{S_q \in \mathbb{S}})$ to $\mathcal{F}_{\mathsf{VSS}}$, where s_q is set to s_{qi} corresponding to any $P_i \in \mathcal{C}_q \setminus Z^\star$.

Figure 7: Simulator for the protocol Π_{PVSS} where Adv corrupts the parties in set $Z^\star \in \mathcal{Z}$.

We now prove a series of claims which will help us prove the theorem. We start with an *honest* P_{D}.

Claim 3.3. If P_{D} is honest, then the view of Adv in the simulated execution of Π_{PVSS} with $\mathcal{S}_{\mathsf{PVSS}}$ is identically distributed to the view of Adv in the real execution of Π_{PVSS} involving honest parties.

Proof. Let $\mathbb{S}^\star \stackrel{def}{=} \{S_q \in \mathbb{S} \mid S_q \cap Z^\star \neq \emptyset\}$. Then the view of Adv during the various executions consists of the following.

- **The shares $\{[s]_q\}_{S_q \in \mathbb{S}^\star}$ distributed by P_{D}**: In the real execution, Adv receives $[s]_q$ from P_{D} for each $S_q \in \mathbb{S}^\star$. In the simulated execution, the simulator provides this to Adv on behalf of P_{D}. Clearly, the distribution of the shares is identical in both the executions.
- **Corresponding to every $S_q \in \mathbb{S}^\star$, messages $(\mathsf{test}, \mathsf{sid}, P_{\mathsf{D}}, q, s_{qj})$ received from party $P_j \in S_q \setminus Z^\star$, as part of pairwise consistency tests, where $s_{qj} = [s]_q$**: While each P_j sends this to Adv in the real execution, the simulator sends this on the behalf of P_j in the simulated execution. Clearly, the distribution of the messages is identical in both the executions.

– **For every $S_q \in \mathbb{S}$ and every $P_i, P_j \in S_q$, the outputs $\mathsf{OK}_q(P_i, \mathsf{Acast}, \mathsf{sid}_{ij}^{(P_\mathsf{D},q)}, \mathsf{OK}_q(i,j))$ of the pairwise consistency tests, received as output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{ij}^{(P_\mathsf{D},q)}$**: To compare the distribution of these messages in the two executions, we consider the following cases, considering an arbitrary $S_q \in \mathbb{S}$ and arbitrary $P_i, P_j \in S_q$.
  – $P_i, P_j \in S_q \setminus Z^\star$: In both the executions, Adv receives $\mathsf{OK}_q(P_i, \mathsf{Acast}, \mathsf{sid}_{ij}^{(P_\mathsf{D},q)}, \mathsf{OK}_q(i,j))$ as the output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{ij}^{(P_\mathsf{D},q)}$.
  – $P_i \in S_q \setminus Z^\star, P_j \in (S_q \cap Z^\star)$: In both the executions, Adv receives $\mathsf{OK}_q(P_i, \mathsf{Acast}, \mathsf{sid}_{ij}^{(P_\mathsf{D},q)}, \mathsf{OK}_q(i,j))$ as the output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{ij}^{(P_\mathsf{D},q)}$ if and only if Adv sent $(\mathsf{test}, \mathsf{sid}, P_\mathsf{D}, q, s_{qj})$ on the behalf of P_j to P_i such that $s_{qj} = [s]_q$ holds.
  – $P_i \in (S_q \cap Z^\star)$: In both the executions, Adv receives $\mathsf{OK}_q(P_i, \mathsf{Acast}, \mathsf{sid}_{ij}^{(q)}, \mathsf{OK}_q(i,j))$ if and only if Adv on the behalf of P_i has sent $(P_i, \mathsf{Acast}, \mathsf{sid}_{ij}^{(P_\mathsf{D},q)}, \mathsf{OK}_q(i,j))$ to $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{ij}^{(P_\mathsf{D},q)}$ for P_j.

  Clearly, irrespective of the case, the distribution of the OK_q messages is identical in both the executions.
– **The core sets $\{\mathcal{C}_q\}_{S_q \in \mathbb{S}}$**: In both the executions, the sets $\mathcal{C}_q$ are determined based on the OK_q messages delivered to P_D. So the distribution of these sets is also identical.

□

We next claim that if the dealer is *honest*, then conditioned on the view of the adversary Adv (which is identically distributed in both the executions, as per the previous claim), the outputs of the honest parties are identically distributed in both the executions.

Claim 3.4. If P_D is honest, then conditioned on the view of Adv, the outputs of the honest parties during the execution of Π_{PVSS} involving Adv has the same distribution as the outputs of the honest parties in the ideal-world involving $\mathcal{S}_{\mathsf{PVSS}}$ and $\mathcal{F}_{\mathsf{VSS}}$.

Proof. Let P_D be honest and let View be an arbitrary view of Adv. Moreover, let $\{s_q\}_{S_q \cap Z^\star \neq \emptyset}$ be the shares of the corrupt parties, as per View. Furthermore, let $\{s_q\}_{S_q \cap Z^\star = \emptyset}$ be the shares used by P_D in the simulated execution, corresponding to the set $S_q \in \mathbb{S}$, such that $S_q \cap Z^\star = \emptyset$. Let $s \stackrel{def}{=} \sum_{S_q \cap Z^\star \neq \emptyset} s_q + \sum_{S_q \cap Z^\star = \emptyset} s_q$. Then in the simulated execution, each *honest* party P_i obtains the output $\{[s]_q\}_{P_i \in S_q}$ from $\mathcal{F}_{\mathsf{VSS}}$, where $[s]_q = s_q$. We now show that P_i eventually obtains the output $\{[s]_q\}_{P_i \in S_q}$ in the real execution as well, if P_D's inputs in the protocol Π_{PVSS} are $\{s_q\}_{S_q \in \mathbb{S}}$.

Since P_D is *honest*, it sends the share s_q to *all* the parties in the set S_q, which is eventually delivered. Now consider an *arbitrary* $S_q \in \mathbb{S}$. During the pairwise consistency tests, each *honest* $P_k \in S_q$ will eventually send $s_{qk} = s_q$ to *all* the parties in S_q. Consequently, every *honest* $P_j \in S_q$ will eventually broadcast the message $\mathsf{OK}_q(j,k)$, corresponding to every *honest* $P_k \in S_q$. This is because $s_{qj} = s_{qk} = s_q$ will hold. These $\mathsf{OK}_q(j,k)$ messages are eventually received by every honest party, including P_D. This implies that the parties in $S_q \setminus Z^\star$ will eventually form a clique in the graph $G_q^{(i)}$ of every *honest* P_i. This further implies that P_D will eventually find a set $\mathcal{C}_q$ where $S_q \setminus \mathcal{C}_q \in \mathcal{Z}$ and where $\mathcal{C}_q$ constitutes a clique in the consistency graph of every honest party. This is because the set $S_q \setminus Z^\star$ is guaranteed to eventually constitute a clique. Hence P_D eventually broadcasts the sets $\{\mathcal{C}_q\}_{S_q \in \mathbb{S}}$, which are eventually delivered to every honest party. Moreover, the verification of these sets will eventually be successful for every honest party.

Next, consider an arbitrary *honest* $P_i \in S_q$. If $P_i \in \mathcal{C}_q$, then it has already received the share s_q from P_D and $s_{qi} = s_q$ holds. Hence, P_i sets $[s]_q$ to s_q. So consider the case when $P_i \notin \mathcal{C}_q$. In this case, P_i sets $[s]_q$ based on the supposedly common values s_{qj} received from the parties $P_j \in S_q$ as part of pairwise consistency tests. Specifically, P_i checks for a subset of parties $\mathcal{C}'_q \subseteq \mathcal{C}_q$, where $\mathcal{C}_q \setminus \mathcal{C}'_q \in \mathcal{Z}$, such that every party $P_j \in \mathcal{C}'_q$ has sent the *same* s_{qj} value to P_i as part of the pairwise consistency test. If P_i finds such a set $\mathcal{C}'_q$, then it sets $[s]_q$ to the common s_{qj}. To complete the proof, we need to show that P_i will eventually find such a set $\mathcal{C}'_q$, and if such a set $\mathcal{C}'_q$ is found by P_i, then the common s_{qj} is the same as s_q.

Assuming that P_i eventually finds such a $\mathcal{C}'_q$, the proof that the common s_{qj} is the same as s_q follows from the fact that $\mathcal{C}'_q$ is guaranteed to contain at least one *honest* party from $\mathcal{C}_q$, who would have received the share $s_{qj} = s_q$ from P_D and sent to P_i as part of the pairwise consistency test. This is because $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition. Also, since the $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition is satisfied, the set of *honest* parties in $\mathcal{C}_q$, namely the parties in $\mathcal{C}_q \setminus Z^\star$, always constitute a candidate $\mathcal{C}'_q$ set. This is because every party $P_j \in \mathcal{C}_q \setminus Z^\star$ would have sent $s_{qj} = s_q$ to every party in S_q during the pairwise consistency test, and these values are eventually delivered. □

We next prove certain claims with respect to a *corrupt* dealer. The first claim is that the view of Adv in this case is also identically distributed in both the real as well as simulated execution. This is simply because in this case, the *honest* parties have *no* inputs and the simulator simply plays the role of the honest parties *exactly* as per the steps of the protocol Π_PVSS in the simulated execution.

Claim 3.5. If P_D is corrupt, then the view of Adv in the simulated execution of Π_PVSS with $\mathcal{S}_\mathsf{PVSS}$ is identically distributed as the view of Adv in the real execution of Π_PVSS involving honest parties.

Proof. The proof follows from the fact that if P_D is *corrupt*, then $\mathcal{S}_\mathsf{PVSS}$ participates in a full execution of the protocol Π_PVSS, by playing the role of the honest parties as per the steps of Π_PVSS. Hence, there is a one-to-one correspondence between simulated executions and real executions. □

We finally claim that if the dealer is *corrupt*, then conditioned on the view of the adversary (which is identical in both the executions as per the last claim), the outputs of the honest parties are identically distributed in both the executions.

Claim 3.6. If D is corrupt, then conditioned on the view of Adv, the output of the honest parties during the execution of Π_PVSS involving Adv has the same distribution as the output of the honest parties in the ideal-world involving $\mathcal{S}_\mathsf{PVSS}$ and $\mathcal{F}_\mathsf{VSS}$.

Proof. Let P_D be *corrupt* and let View be an arbitrary view of Adv. We note that whether valid core sets $\{\mathcal{C}_q\}_{S_q \in \mathbb{S}}$ have been generated during the corresponding execution of Π_PVSS or not can be found out from View. We now consider the following cases.

- *No core sets $\{\mathcal{C}_q\}_{S_q \in \mathbb{S}}$ are generated as per* View: In this case, the honest parties do not obtain any output in either execution. This is because in the real execution of Π_PVSS, the honest parties compute their output only when they get valid core sets $\{\mathcal{C}_q\}_{S_q \in \mathbb{S}}$ from P_D's broadcast. If this is not the case, then in the simulated execution, the simulator $\mathcal{S}_\mathsf{PVSS}$ does not provide any input to $\mathcal{F}_\mathsf{VSS}$ on behalf of P_D; hence, $\mathcal{F}_\mathsf{VSS}$ does not produce any output for the honest parties.
- *Core sets $\{\mathcal{C}_q\}_{S_q \in \mathbb{S}}$ generated as per* View *are invalid*: Again, in this case, the honest parties do not obtain any output in either execution. This is because in the real execution of Π_PVSS, even if the sets $\{\mathcal{C}_q\}_{S_q \in \mathbb{S}}$ are received from P_D's broadcast, the honest parties compute their output only when each set $\mathcal{C}_q$ is found to be *valid* with respect to the verifications performed by the honest parties in their own consistency graphs. If these verifications fail (implying that the core sets are invalid), then in the simulated execution, the simulator $\mathcal{S}_\mathsf{PVSS}$ does not provide any input to $\mathcal{F}_\mathsf{VSS}$ on behalf of P_D, implying that $\mathcal{F}_\mathsf{VSS}$ does not produce any output for the honest parties.
- *Valid core sets $\{\mathcal{C}_q\}_{S_q \in \mathbb{S}}$ are generated as per* View: We first note that in this case, P_D has distributed some common share, say s_q, determined by View, to all the parties in $\mathcal{C}_q \setminus Z^\star$ during the real execution of Π_PVSS. This is because all the parties in $\mathcal{C}_q \setminus Z^\star$ are *honest*, and form a clique in the consistency

graph of the honest parties. Hence, each $P_j, P_k \in \mathcal{C}_q \setminus Z^\star$ has broadcasted the messages $\mathsf{OK}_q(j,k)$ and $\mathsf{OK}_q(k,j)$ after checking that $s_{qj} = s_{qk}$ holds, where s_{qj} and s_{qk} are the shares received from P_D by P_j and P_k respectively.

We next show that in the real execution of Π_{PVSS}, every party in $S_q \setminus Z^\star$, eventually sets $[s]_q = s_q$. While this is true for the parties in $\mathcal{C}_q \setminus Z^\star$, we consider an arbitrary party $P_i \in S_q \setminus (Z^\star \cup \mathcal{C}_q)$. From the protocol steps, P_i checks for a subset of parties $\mathcal{C}'_q \subseteq \mathcal{C}_q$ where $\mathcal{C}_q \setminus \mathcal{C}'_q \in \mathcal{Z}$, such that every party $P_j \in \mathcal{C}'_q$ has sent the *same* s_{qj} value to P_i as part of the pairwise consistency test. If P_i finds such a set $\mathcal{C}'_q$, then it sets $[s]_q$ to the common s_{qj}. We next argue that P_i will eventually find such a set $\mathcal{C}'_q$ and if such a set $\mathcal{C}'_q$ is found by P_i, then the common s_{qj} is the same as s_q. The proof for this is exactly the *same*, as for Claim 3.4.

Thus, in the real execution, every honest party P_i eventually outputs $\{[s]_q = s_q\}_{P_i \in S_q}$. From the steps of $\mathcal{S}_{\mathsf{PVSS}}$, the simulator sends the shares $\{s_q\}_{S_q \in \mathbb{S}}$ to $\mathcal{F}_{\mathsf{VSS}}$ on the behalf of P_D in the simulated execution. Consequently, in the ideal world, $\mathcal{F}_{\mathsf{VSS}}$ will eventually deliver the shares $\{[s]_q = s_q\}_{P_i \in S_q}$ to every honest P_i. Hence, the outputs of the honest parties are identical in both worlds.

□

The proof of the theorem now follows from Claims 3.3-3.6. □

Computational Complexity of the Protocol Π_{PVSS}. In Π_{PVSS}, the most computationally-expensive step is to compute the core sets. This can always be done with computation complexity $\mathcal{O}(\text{poly}(n, |\mathcal{Z}|))$. For instance, to check for the existence of a core set $\mathcal{C}_q$, the dealer P_D can do the following after every update in its consistency graph G_q^D: for every $Z_i \in \mathcal{Z}$, check if there exists an edge in the graph G_q^D, between every pair of parties in $S_q \setminus Z_i$. This may require iterating over the entire $\mathcal{Z}$ and within each iteration, checking for the presence of a clique will further require $\mathcal{O}(\text{poly}(n))$ computational effort. Since there can be $\mathcal{O}(n^2)$ updates in the graph G_q^D, the existence of $\mathcal{C}_q$ can be verified with $\mathcal{O}(\text{poly}(n, |\mathcal{Z}|))$ computational effort.

Reducing the Broadcast Complexity of the Protocol Π_{PVSS}. Protocol Π_{PVSS}, as presented in Fig 6, has a *broadcast complexity* proportional to the size of $\mathcal{Z}$. More specifically, in the protocol, P_D needs to compute a core set $\mathcal{C}_q$ corresponding to each $S_q \in \mathbb{S}$. For finding these core sets, every (honest) party needs to broadcast an OK_q message for every other (honest) party by calling $\mathcal{F}_{\mathsf{Acast}}$. This results in the number of bits broadcasted being proportional to $|\mathbb{S}|$, where $|\mathbb{S}| = |\mathcal{Z}|$ in our case. A small modification to the protocol can make the broadcast complexity *independent* of $|\mathcal{Z}|$. The idea is to let every party broadcast a *single* OK message for every other party if the pairwise consistency test with that party is successful across *all* the sets S_q to which both the parties belong. In more detail, party P_i sends an $\mathsf{OK}(i,j)$ message to $\mathcal{F}_{\mathsf{Acast}}$, only after checking whether $s_{qi} = s_{qj}$ holds corresponding to *every* $S_q \in \mathbb{S}$, such that $P_j \in S_q$ holds. Consequently, P_D now checks for the presence of a *single* core set $\mathcal{C}$, such that all the following hold.[9]

- For every $P_i, P_j \in \mathcal{C}$, the messages $\mathsf{OK}(i,j)$ and $\mathsf{OK}(j,i)$ have been received from the corresponding $\mathcal{F}_{\mathsf{Acast}}$ instances.
- For $q = 1, \ldots, |\mathbb{S}|$, $\mathcal{C} \subseteq S_q$
- For $q = 1, \ldots, |\mathbb{S}|$, $S_q \setminus \mathcal{C} \in \mathcal{Z}$.

Upon finding such a $\mathcal{C}$, the dealer broadcasts it by sending it to $\mathcal{F}_{\mathsf{Acast}}$. Note that such a set $\mathcal{C}$ is eventually obtained for an *honest* P_D. This is because the set of parties $(S_1 \setminus Z^\star) \cap \ldots \cap (S_q \setminus Z^\star)$ constitutes a candidate $\mathcal{C}$ for an honest P_D, where $Z^\star$ is the set of *corrupt* parties. The rest of the steps of the protocol remain the same. With these modifications, the communication complexity of the protocol Π_{PVSS} is computed as follows. The dealer needs to send the share $s^{(q)}$ to all the parties in S_q, and every party in S_q has to send the received share to every other party in S_q during pairwise consistency tests. This incurs a communication

[9] Checking for the existence of such a $\mathcal{C}$ can be always performed with $\mathcal{O}(\text{poly}(n, |\mathcal{Z}|))$ computational effort.

of $\mathcal{O}(|\mathcal{Z}| \cdot n^2 \log |\mathbb{F}|)$ bits, since each $|S_q| = \mathcal{O}(n)$ and each share $s^{(q)}$ can be represented by $\log |\mathbb{F}|$ bits. There will be total $\mathcal{O}(n^2)$ OK messages broadcasted, where each message can be represented by $\mathcal{O}(\log n)$ bits, since it represents the index of two parties. Moreover, P_{D} will broadcast a single core set $\mathcal{C}$ of size $\mathcal{O}(n \log n)$ bits. Based on this discussion, we next state the following theorem for the modified version of Π_{PVSS}.

Theorem 3.7. *Consider a static malicious adversary* Adv *characterized by an adversary-structure* $\mathcal{Z}$*, satisfying the* $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ *condition and let* $\mathbb{S} = \{S_1, \ldots, S_h\} \stackrel{def}{=} \{\mathcal{P} \setminus Z | Z \in \mathcal{Z}\}$ *be the sharing specification. Then protocol* Π_{PVSS} *UC-securely realizes the functionality* $\mathcal{F}_{\mathsf{VSS}}$ *with perfect security in the* $\mathcal{F}_{\mathsf{Acast}}$*-hybrid model, in the presence of* Adv*. The protocol makes* $\mathcal{O}(n^2)$ *calls to* $\mathcal{F}_{\mathsf{Acast}}$ *with* $\mathcal{O}(\log n)$ *bit messages, one call to* $\mathcal{F}_{\mathsf{Acast}}$ *with* $\mathcal{O}(n \log n)$ *bit message and additionally incurs a communication of* $\mathcal{O}(|\mathcal{Z}| \cdot n^2 \log |\mathbb{F}|)$ *bits.*

By replacing the calls to $\mathcal{F}_{\mathsf{Acast}}$ *with protocol* Π_{Acast}*, the protocol incurs a total communication of* $\mathcal{O}(|\mathcal{Z}| \cdot n^2 \log |\mathbb{F}| + n^4 \log n)$ *bits.*

3.1.1 Asynchronous Reconstruction Protocols

Let s be a value which is secret-shared with respect to some sharing specification $\mathbb{S} = \{S_1, \ldots, S_{|\mathbb{S}|}\}$, such that $\mathbb{S}$ satisfies the $\mathbb{Q}^{(2)}(\mathbb{S}, \mathcal{Z})$ condition. We first present a protocol $\Pi_{\mathsf{PerRecShare}}$ which allows *all* parties in $\mathcal{P}$ to reconstruct a *single* share $[s]_q$ corresponding to any designated set $S_q \in \mathbb{S}$. In the protocol, every party in S_q sends the share $[s]_q$ to all the parties outside S_q, which then "filter" out the potentially incorrect versions of $[s]_q$ and output $[s]_q$. Protocol $\Pi_{\mathsf{PerRecShare}}$ is formally presented in Figure 8.

Protocol $\Pi_{\mathsf{PerRecShare}}(q)$

- **Sending Share to All Parties**: If $P_i \in S_q$, then execute the following steps.
  1. On having the share $[s]_q$, send $(\mathsf{share}, \mathsf{sid}, q, [s]_q)$ to all the parties in $\mathcal{P} \setminus S_q$.
- **Computing Output**: Based on the following conditions, execute the corresponding steps.
  1. $P_i \in S_q$: Output $[s]_q$.
  2. $P_i \notin S_q$: Upon receiving $(\mathsf{share}, \mathsf{sid}, q, v)$ from a set of parties $S'_q \subseteq S_q$ such that $S_q \setminus S'_q \in \mathcal{Z}$, output $[s]_q = v$.

Figure 8: Perfectly-secure reconstruction protocol for session id sid to publicly reconstruct the share $[s]_q$ corresponding to $S_q \in \mathbb{S}$. The public inputs are $\mathcal{P}$, $\mathcal{Z}$ and $\mathbb{S}$. The above steps are executed by every $P_i \in \mathcal{P}$

Lemma 3.8. *Let* $\mathcal{Z}$ *be an adversary structure and let* $\mathbb{S} = \{S_1, \ldots, S_{|\mathbb{S}|}\}$ *be a sharing specification, such that* $\mathbb{S}$ *satisfies the* $\mathbb{Q}^{(2)}(\mathbb{S}, \mathcal{Z})$ *condition. Moreover, let* s *be a value, which is secret-shared as per* $\mathbb{S}$*. Then for any* $q \in \{1, \ldots, |\mathbb{S}|\}$ *and any adversary* Adv *corrupting a set of parties* $Z^\star \in \mathcal{Z}$*, all honest parties eventually output the share* $[s]_q$ *in the protocol* $\Pi_{\mathsf{PerRecShare}}$*. The protocol incurs a communication of* $\mathcal{O}(n^2 \log |\mathbb{F}|)$ *bits.*

Proof. Consider an arbitrary *honest* party $P_i \in \mathcal{P}$. We consider two cases.

- $P_i \in S_q$: In this case, P_i outputs $[s]_q$.
- $P_i \notin S_q$: In this case, P_i waits for a subset of parties $S'_q \subseteq S_q$ where $S_q \setminus S'_q \in \mathcal{Z}$, such that every party $P_j \in S'_q$ has sent the *same* share v to P_i. If P_i finds such a set S'_q, then it outputs v. To complete the proof, we need to show that P_i will eventually find such a set S'_q and if such a set S'_q is found by P_i, then the common value v is the same as $[s]_q$.

Assuming that P_i eventually finds such a common S'_q, the proof that the common value v is the same as $[s]_q$ follows from the fact that S'_q is guaranteed to contain at least one *honest* party from S_q, who would have sent the share $[s]_q$ to P_i. This is because the $\mathbb{Q}^{(2)}(\mathbb{S}, \mathcal{Z})$ condition is satisfied. Also, since the $\mathbb{Q}^{(2)}(\mathbb{S}, \mathcal{Z})$ condition is satisfied, the set of *honest* parties in S_q, namely the parties in $S_q \setminus Z^\star$, always constitute a candidate S'_q set. This is because every party $P_j \in S_q \setminus Z^\star$ would have sent $[s]_q$ to P_i and these values are eventually delivered to P_i.

The communication complexity follows from the protocol steps. □

We now present the protocol Π_{PerRec} (Fig 9), which allows *all* parties in $\mathcal{P}$ to reconstruct a secret shared value s. The idea is to run an instance of $\Pi_{\mathsf{PerRecShare}}$ for each $S_q \in \mathbb{S}$, and to sum up the shares obtained as the output from each instance.

Protocol Π_{PerRec}

- **Reconstructing Shares**: For each $S_q \in \mathbb{S}$, participate in an instance $\Pi_{\mathsf{PerRecShare}}(q)$ with sid to obtain the output $[s]_q$.
- **Output Computation**: Output $s = \sum_{S_q \in \mathbb{S}} [s]_q$.

Figure 9: Perfectly-secure reconstruction protocol for session id sid to reconstruct a shared value s. The public inputs of the protocol are $\mathcal{P}, \mathbb{S}$ and $\mathcal{Z}$. The above steps are executed by every $P_i \in \mathcal{P}$

The properties of the protocol Π_{PerRec} are stated in Lemma 3.9, which follow from the protocol steps and Lemma 3.8.

Lemma 3.9. *Let $\mathcal{Z}$ be an adversary structure and let $\mathbb{S} = \{S_1, \ldots, S_{|\mathbb{S}|}\}$ be a sharing specification, such that $\mathbb{S}$ satisfies the $\mathbb{Q}^{(2)}(\mathbb{S}, \mathcal{Z})$ condition. Moreover, let s be a value which is secret-shared as per $\mathbb{S}$. Then for every adversary Adv corrupting a set of parties $Z^\star \in \mathcal{Z}$, all honest parties eventually output s in the protocol $\Pi_{\mathsf{PerRecShare}}$. The protocol incurs a communication of $\mathcal{O}(|\mathbb{S}| \cdot n^2 \log |\mathbb{F}|)$ bits, which is $\mathcal{O}(|\mathcal{Z}| \cdot n^2 \log |\mathbb{F}|)$ bits if $|\mathbb{S}| = |\mathcal{Z}|$.*

3.2 Perfectly-Secure Multiplication Protocol

We next present a perfectly-secure multiplication protocol which takes input $\{([a^{(\ell)}], [b^{(\ell)}])\}_{\ell=1,\ldots,M}$ and outputs $\{[c^{(\ell)}]\}_{\ell=1,\ldots,M}$, where $c^{(\ell)} = a^{(\ell)} b^{(\ell)}$, without revealing any additional information about $\{a^{(\ell)}, b^{(\ell)}\}_{\ell=1,\ldots,M}$. We first explain and present the protocol assuming $M = 1$, where the inputs are $[a]$ and $[b]$ and the goal is to securely generate a random sharing $[ab]$ of ab. The modifications to handle M pairs of inputs are straightforward.

We briefly recall the high-level idea behind our multiplication protocol which had been discussed in detail in Section 1.1. We first design an *asynchronous* optimistic multiplication protocol Π_{OptMult} which takes as input a set $Z \in \mathcal{Z}$ and generates a secret-sharing of ab, *provided* Adv corrupts a set of parties $Z^\star \subseteq Z$. Using protocol Π_{OptMult}, the parties then proceed in *iterations*, where in each iteration, the parties run an instance of the *asynchronous* Π_{OptMult} for each $Z \in \mathcal{Z}$. They then compare the outputs from each instance to detect if the corrupt parties cheated in any of the instances, and proceed to the respective cheater-identification phase if any cheating is detected. An iteration "fails" if cheating is detected in the form of a pair of "conflicting" Π_{OptMult} instances, where the resultant secret-shared outputs are *different*. If this happens, then the parties temporarily "wait-list" all the parties who have shared any summand during the conflicting instances of Π_{OptMult} for that iteration. The summand-sharing parties stay on the waiting-list till they complete all their supposed tasks in the corresponding cheater-identification phase, after which they are "released" to participate in instances of Π_{OptMult} in future iterations. This mechanism ensures that if an iteration fails, then the cheating parties from that iteration cannot participate in future iterations till they participate in the pending cheater identification phase of the failed iteration, in which case they are eventually discarded by all honest parties. This process is repeated till the parties reach a "successful" iteration where no cheating is detected (where the outputs of all the Π_{OptMult} instances are the same). We will show that there will be *at most* $t(tn + 1) + 1$ iterations within which a successful iteration is reached, where t is the cardinality of the maximum-sized subset in $\mathcal{Z}$.

Based on the above discussion, we next present protocols $\Pi_{\mathsf{OptMult}}, \Pi_{\mathsf{MultCI}}$ and Π_{Mult}. Protocol Π_{MultCI} (multiplication with cheater-identification) represents an iteration as discussed above. In the protocol, the parties run an instance of Π_{OptMult} for each $Z \in \mathcal{Z}$. If a pair of conflicting Π_{OptMult} instances with different outputs are identified, then the parties proceed to execute the corresponding cheater-identification phase. Protocol Π_{Mult} iteratively calls Π_{MultCI} multiple times till it reaches a "successful" instance of Π_{MultCI}, where the outputs of all the instances of Π_{OptMult} are the same. Across all the instances of these protocols, the parties maintain the following *dynamic* sets:

- $\mathcal{W}^{(i)}_{\mathsf{iter}}$: Denotes the parties *wait-listed* by P_i corresponding to instance number iter of Π_{MultCI} during Π_{Mult}. If P_i detects any cheating during the instance number iter of Π_{MultCI} with a pair of conflicting Π_{OptMult} instances, then all the summand-sharing parties from the conflicting instances are included in $\mathcal{W}^{(i)}_{\mathsf{iter}}$. These parties are removed from $\mathcal{W}^{(i)}_{\mathsf{iter}}$ as and when they execute their respective steps of the corresponding cheater-identification phase.
- $\mathcal{LD}^{(i)}_{\mathsf{iter}}$: The set of parties from $\mathcal{W}^{(i)}_{\mathsf{iter}}$ which are *locally discarded* by P_i during the cheater-identification phase of instance number iter of Π_{MultCI} in Π_{Mult}.
- $\mathcal{GD}$: Denotes the set of parties, *globally discarded* by *all* (honest) parties across various instances of Π_{MultCI} in protocol Π_{Mult}.[10]

Looking ahead, these sets will be maintained in such a way that no honest party is ever included in the $\mathcal{GD}$ and $\mathcal{LD}^{(i)}_{\mathsf{iter}}$ sets of any honest P_i. Moreover, any honest party which is included in the $\mathcal{W}^{(i)}_{\mathsf{iter}}$ set of any honest P_i will eventually be removed. Consequently, it will be ensured that each honest party is allowed to eventually participate in all the instances of Π_{OptMult} and hence all the instances of Π_{OptMult} eventually produce some output and never get stuck forever.

3.2.1 Optimistic Multiplication Protocol

Protocol Π_{OptMult} is executed with respect to a given $Z \in \mathcal{Z}$ and iteration number iter. The inputs of the protocol are $[a]$ and $[b]$. The protocol is *guaranteed* to eventually generate an output, which will be $[ab]$ if no party *outside* the set Z behaves maliciously. The idea behind the protocol is as follows. Since $ab = \sum_{(p,q) \in \{1,\ldots,|\mathbb{S}|\} \times \{1,\ldots,|\mathbb{S}|\}} [a]_p [b]_q$, a secret-sharing of ab can be computed *locally* from secret-sharing of the summands $[a]_p[b]_q$, owing to the linearity property of the secret-sharing. If $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition, each $(S_p \cap S_q) \setminus Z$ contains at least one *honest* party. Since the parties may not know the identity of the honest parties in the set $(S_p \cap S_q) \setminus Z$, *every* party in $(S_p \cap S_q) \setminus Z$ tries to secret-share the summand $[a]_p[b]_q$. For the sake of efficiency, instead of sharing a single summand, each party in $\mathcal{P} \setminus Z$ tries to act as a summand-sharing party and shares the sum of all the summands it is "capable" of. To ensure that each summand $[a]_p[b]_q$ is secret-shared exactly by one party, the parties select *distinct* summand-sharing parties in hops. The summands whose sum has been shared by the elected party are "marked" as shared, ensuring that they are not considered in future hops. To agree on the summand-sharing party of each hop, the parties execute an instance of the *agreement on common subset* (ACS) primitive [4], where one instance of ABA is invoked on the behalf of each candidate summand-sharing party. While voting for a candidate party from

[10] The reason for two different discarded sets is that the various instances of cheater-identification corresponding to the failed Π_{MultCI} instances are executed *asynchronously*, thus resulting in a corrupt party to be identified by different honest parties during different iterations.

$\mathcal{P} \setminus Z$ during a hop, the parties ensure that the candidate has indeed secret-shared some sum and additionally satisfies the following conditions:

– The candidate party has not been selected in an earlier hop.
– The candidate party does not belong to the waiting list or the list of locally-discarded parties of any previous iteration.
– The candidate does not belong to the list of globally-discarded parties.

Protocol $\Pi_{\mathsf{OptMult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], Z, \mathsf{iter})$

– **Initialization**
  - Initialize *summand-index-set* of indices of *all* summands : $\mathsf{SIS}_{(Z,\mathsf{iter})} = \{(p,q)\}_{p,q=1,\ldots,|\mathbb{S}|}$.
  - Initialize the *summand-index-set* corresponding to $P_j \in \mathcal{P} \setminus Z$: $\mathsf{SIS}^{(j)}_{(Z,\mathsf{iter})} = \{(p,q)\}_{P_j \in S_p \cap S_q}$.
  - Initialize the set of summands-sharing parties : $\mathsf{Selected}_{(Z,\mathsf{iter})} = \emptyset$.
  - Initialize the hop number $\mathsf{hop} = 1$.

– While $\mathsf{SIS}_{(Z,\mathsf{iter})} \neq \emptyset$, do the following:
  - **Sharing Sum of Summands**:
    1. If $P_i \notin Z$ and $P_i \notin \mathsf{Selected}_{(Z,\mathsf{iter})}$, then compute $c^{(i)}_{(Z,\mathsf{iter})} = \sum_{(p,q) \in \mathsf{SIS}^{(i)}_{(Z,\mathsf{iter})}} [a]_p[b]_q$. Randomly select the shares $c^{(i)}_{{(Z,\mathsf{iter})}_1}, \ldots, c^{(i)}_{{(Z,\mathsf{iter})}_h}$, such that $c^{(i)}_{{(Z,\mathsf{iter})}_1} + \ldots + c^{(i)}_{{(Z,\mathsf{iter})}_h} = c^{(i)}_{(Z,\mathsf{iter})}$. Call $\mathcal{F}_{\mathsf{VSS}}$ with $(\mathsf{dealer}, \mathsf{sid}_{\mathsf{hop},i,\mathsf{iter},Z}, (c^{(i)}_{{(Z,\mathsf{iter})}_1}, \ldots, c^{(i)}_{{(Z,\mathsf{iter})}_h}))$, where $\mathsf{sid}_{\mathsf{hop},i,\mathsf{iter},Z} = \mathsf{hop}||\mathsf{sid}||i||\mathsf{iter}||Z$.[a]
    2. Keep requesting for an output from $\mathcal{F}_{\mathsf{VSS}}$ with $\mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}$, corresponding to every $P_j \in \mathcal{P} \setminus Z$, till an output is received.
  - **Selecting Summand-Sharing Party Through ACS**:
    1. For $j = 1, \ldots, n$, send $(\mathsf{vote}, \mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}, 1)$ to $\mathcal{F}_{\mathsf{ABA}}$, if *all* the following conditions hold:
       – $P_j \notin \mathcal{GD}$;
       – $P_j \notin Z$;
       – $P_j \notin \mathsf{Selected}_{(Z,\mathsf{iter})}$;
       – $\forall \mathsf{iter}' < \mathsf{iter}$, $P_j \notin \mathcal{W}^{(i)}_{\mathsf{iter}'}$ and $P_j \notin \mathcal{LD}^{(i)}_{\mathsf{iter}'}$;
       – An output $(\mathsf{share}, \mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}, P_j, \{[c^{(j)}_{(Z,\mathsf{iter})}]_q\}_{P_i \in S_q})$ is received from $\mathcal{F}_{\mathsf{VSS}}$, with $\mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}$.
    2. For $j = 1, \ldots, n$, request for an output from $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}$, until an output is received.
    3. Upon receiving $(\mathsf{decide}, \mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}, 1)$ from $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}$ corresponding to any $P_j \in \mathcal{P}$, corresponding to each $P_k \in \mathcal{P}$ for which no vote message has been sent yet, send $(\mathsf{vote}, \mathsf{sid}_{\mathsf{hop},k,\mathsf{iter},Z}, 0)$ to $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{\mathsf{hop},k,\mathsf{iter},Z}$.
    4. Once an output $(\mathsf{decide}, \mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}, v_j)$ is received from $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}$ for all $j \in \{1, \ldots, n\}$, select the least indexed P_j, such that $v_j = 1$. Then set $\mathsf{hop} = \mathsf{hop} + 1$ and update the following.
       – $\mathsf{Selected}_{(Z,\mathsf{iter})} = \mathsf{Selected}_{(Z,\mathsf{iter})} \cup \{P_j\}$.
       – $\mathsf{SIS}_{(Z,\mathsf{iter})} = \mathsf{SIS}_{(Z,\mathsf{iter})} \setminus \mathsf{SIS}^{(j)}_{(Z,\mathsf{iter})}$.
       – $\forall P_k \in \mathcal{P} \setminus \{Z \cup \mathsf{Selected}_{(Z,\mathsf{iter})}\}$: $\mathsf{SIS}^{(k)}_{(Z,\mathsf{iter})} = \mathsf{SIS}^{(k)}_{(Z,\mathsf{iter})} \setminus \mathsf{SIS}^{(j)}_{(Z,\mathsf{iter})}$.

– $\forall P_j \in \mathcal{P} \setminus \mathsf{Selected}_{(Z,\mathsf{iter})}$, participate in an instance of Π_{PerDefSh} with input $c^{(j)}_{(Z,\mathsf{iter})} = 0$.

– Output $\{[c^{(1)}_{(Z,\mathsf{iter})}]_q, \ldots, [c^{(n)}_{(Z,\mathsf{iter})}]_q, [c_{(Z,\mathsf{iter})}]_q\}_{P_i \in S_q}$, where $c_{(Z,\mathsf{iter})} \stackrel{def}{=} c^{(1)}_{(Z,\mathsf{iter})} + \ldots + c^{(n)}_{(Z,\mathsf{iter})}$.

[a] The notation $\mathsf{sid}_{\mathsf{hop},i,\mathsf{iter},Z}$ is used to distinguish among the different calls to $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$ within each hop.

Figure 10: Optimistic multiplication in $(\mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{ABA}})$-hybrid for iteration iter and session id sid, assuming Z to be corrupt. The above code is executed by each P_i, who implicitly uses the dynamic sets $\mathcal{GD}$, $\mathcal{W}^{(i)}_{\mathsf{iter}'}$ and $\mathcal{LD}^{(i)}_{\mathsf{iter}'}$ for all $\mathsf{iter}' < \mathsf{iter}$

We next formally prove the properties of the protocol Π_{OptMult}. While proving these properties, we will assume that $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition. This further implies that the sharing specification $\mathbb{S} = \{S_1, \ldots, S_h\} \stackrel{def}{=} \{\mathcal{P} \setminus Z | Z \in \mathcal{Z}\}$ satisfies the $\mathbb{Q}^{(3)}(\mathbb{S}, \mathcal{Z})$ condition. Moreover, we assume that for every iter, the following conditions hold.

- No honest party is ever included in the set $\mathcal{GD}$;
- All honest parties are eventually removed from the $\mathcal{W}^{(i)}_{\mathsf{iter}'}, \mathcal{LD}^{(i)}_{\mathsf{iter}'}$ sets of every honest P_i for every $\mathsf{iter}' < \mathsf{iter}$

Looking ahead, these conditions are guaranteed in the protocols Π_{MultCI} and Π_{Mult} (where these sets are constructed and managed), where Π_{OptMult} is used as a subprotocol (see Lemma 3.27 and Claim 3.31 later).

Claim 3.10. For every $Z \in \mathcal{Z}$ and every ordered pair $(p, q) \in \{1, \ldots, |\mathbb{S}|\} \times \{1, \ldots, |\mathbb{S}|\}$, the set $(S_p \cap S_q) \setminus Z$ contains at least one honest party.

Proof. From the definition of the sharing specification $\mathbb{S}$, we have $S_p = \mathcal{P} \setminus Z_p$ and $S_q = \mathcal{P} \setminus Z_q$, where $Z_p, Z_q \in \mathcal{Z}$. Let $Z^\star \in \mathcal{Z}$ be the set of corrupt parties during the protocol Π_{OptMult}. If $(S_p \cap S_q) \setminus Z$ *does not* contain any honest party, then it implies that $((S_p \cap S_q) \setminus Z) \subseteq Z^\star$. This further implies that $\mathcal{P} \subseteq Z_p \cup Z_q \cup Z \cup Z^\star$, implying that $\mathcal{Z}$ *does not* satisfy the $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition, which is a contradiction. ☐

Claim 3.11. For every $Z \in \mathcal{Z}$, if all honest parties participate during the hop number hop in the protocol Π_{OptMult}, then all honest parties eventually obtain a common summand-sharing party, say P_j, for this hop, such that the honest parties will eventually hold $[c^{(j)}_{(Z,\mathsf{iter})}]$. Moreover, party P_j will be distinct from the summand-sharing party selected for any hop number $\mathsf{hop}' < \mathsf{hop}$.

Proof. Since all honest parties participate in hop number hop, it follows that $\mathsf{SIS}_{(Z,\mathsf{iter})} \neq \emptyset$ at the beginning of hop number hop. This implies that there exists at least one ordered pair $(p, q) \in \mathsf{SIS}_{(Z,\mathsf{iter})}$. From Claim 3.10, there exists at least one *honest* party in $(S_p \cap S_q) \setminus Z$, say P_k, who will have both the shares $[a]_p$ as well as $[b]_q$ (and hence the summand $[a]_p[b]_q$). We also note that P_k *would not* have been selected as the common summand-sharing party in any previous $\mathsf{hop}' < \mathsf{hop}$, as otherwise, P_k would have already included the summand $[a]_p[b]_q$ in the sum $c^{(k)}_{(Z,\mathsf{iter})}$ shared by P_k during hop number hop', implying that $(p, q) \notin \mathsf{SIS}_{(Z,\mathsf{iter})}$. Now, during the hop number hop, party P_k will randomly secret-share the sum $c^{(k)}_{(Z,\mathsf{iter})}$ by making a call to $\mathcal{F}_{\mathsf{VSS}}$, and every honest P_i will eventually receive an output $(\mathsf{share}, \mathsf{sid}_{\mathsf{hop},k,\mathsf{iter},Z}, P_k, \{[c^{(k)}_{(Z,\mathsf{iter})}]_q\}_{P_i \in S_q})$ from $\mathcal{F}_{\mathsf{VSS}}$ with $\mathsf{sid}_{\mathsf{hop},k,\mathsf{iter},Z}$. Moreover, P_k *will not* be present in the set $\mathcal{GD}$ and if P_k is present in the sets $\mathcal{W}^{(i)}_{\mathsf{iter}'}, \mathcal{LD}^{(i)}_{\mathsf{iter}'}$ of any honest P_i for any $\mathsf{iter}' < \mathsf{iter}$, then will eventually be removed from these sets.[11] We next claim that during the hop number hop, there will be at least one instance of $\mathcal{F}_{\mathsf{ABA}}$ corresponding to which *all* honest parties eventually receive the output 1. For this, we consider two possible cases:

- *At least one honest party participates with input* 0 *in the* $\mathcal{F}_{\mathsf{ABA}}$ *instance corresponding to* P_k: Let P_i be an *honest* party, who sends $(\mathsf{vote}, \mathsf{sid}_{\mathsf{hop},k,\mathsf{iter},Z}, 0)$ to $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{\mathsf{hop},k,\mathsf{iter},Z}$. Then from the steps of Π_{OptMult}, it follows that there exists some $P_j \in \mathcal{P}$, such that P_i has received $(\mathsf{decide}, \mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}, 1)$ as the output from $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}$. Hence, *every* honest party will eventually receive the output $(\mathsf{decide}, \mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}, 1)$ as the output from $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}$.
- *No honest party participates with input* 0 *in the* $\mathcal{F}_{\mathsf{ABA}}$ *instance corresponding to* P_k: In this case, *every* honest party will eventually send $(\mathsf{vote}, \mathsf{sid}_{\mathsf{hop},k,\mathsf{iter},Z}, 1)$ to $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{\mathsf{hop},k,\mathsf{iter},Z}$ and eventually receives the output $(\mathsf{decide}, \mathsf{sid}_{\mathsf{hop},k,\mathsf{iter},Z}, 1)$ from $\mathcal{F}_{\mathsf{ABA}}$.

Now, based on the above claim, we can further claim that all honest parties will eventually participate with some input in all the n instances of $\mathcal{F}_{\mathsf{ABA}}$ invoked during the hop number hop and hence, all the n instances

[11] Recall that we are assuming that no honest party is ever included in the set $\mathcal{GD}$ and all honest parties are eventually removed from the $\mathcal{W}^{(i)}_{\mathsf{iter}'}, \mathcal{LD}^{(i)}_{\mathsf{iter}'}$ sets of every honest P_i for every $\mathsf{iter}' < \mathsf{iter}$.

of $\mathcal{F}_{\mathsf{ABA}}$ during the hop number hop will eventually produce an output. Since the summand-sharing party for hop number hop corresponds to the least indexed $\mathcal{F}_{\mathsf{ABA}}$ instance in which all the honest parties obtain 1 as the output, it follows that eventually, the honest parties will select a summand-sharing party. Moreover, this summand-sharing party will be common, as it is based on the outcome of $\mathcal{F}_{\mathsf{ABA}}$ instances.

Let P_j be the summand-sharing party for the hop number hop. We next show that the honest parties will eventually hold $[c^{(j)}_{(Z,\mathsf{iter})}]$. For this, we note that since P_j has been selected as the summand-sharing party, *at least* one *honest* party, say P_i, must have sent $(\mathsf{vote}, \mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}, 1)$ to $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}$. If not, then $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}$ will never produce the output $(\mathsf{decide}, \mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}, 1)$ and hence, P_j will not be the summand-sharing party for the hop number hop. Now since P_i sent $(\mathsf{vote}, \mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}, 1)$ to $\mathcal{F}_{\mathsf{ABA}}$, it follows that P_i has received an output $(\mathsf{share}, \mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}, P_j, \{[c^{(j)}_{(Z,\mathsf{iter})}]_q\}_{P_i \in S_q})$ from $\mathcal{F}_{\mathsf{VSS}}$ with $\mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}$. This implies that P_j must have sent the message $(\mathsf{dealer}, \mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}, (c^{(j)}_{(\mathsf{iter},Z)_1}, \ldots, c^{(j)}_{(\mathsf{iter},Z)_h}))$ to $\mathcal{F}_{\mathsf{VSS}}$ with $\mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}$. Consequently, every honest party will eventually receive their respective outputs from $\mathcal{F}_{\mathsf{VSS}}$ with $\mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}$ and hence, the honest parties will eventually hold $[c^{(j)}_{(Z,\mathsf{iter})}]$.

Finally, to complete the proof of the claim, we need to show that party P_j is different from the summand-sharing parties selected during the hops $1, \ldots, \mathsf{hop} - 1$. If P_j has been selected as a summand-sharing party for any hop number $\mathsf{hop}' < \mathsf{hop}$, then no honest party ever sends $(\mathsf{vote}, \mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}, 1)$ to $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}$. Consequently, $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}$ will never send the output $(\mathsf{decide}, \mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}, 1)$ to any honest party and hence P_j will not be selected as the summand-sharing party for hop number hop, which is a contradiction. □

Claim 3.12. In protocol Π_{OptMult}, all honest parties eventually obtain an output. The protocol makes $\mathcal{O}(n^2)$ calls to $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$.

Proof. From Claim 3.10 and 3.11, it follows that the number of hops in the protocol is $\mathcal{O}(n)$, as in each hop a new summand-sharing party is selected and if all honest parties are included in the set of summand-sharing parties $\mathsf{Selected}_{(Z,\mathsf{iter})}$, then $\mathsf{SIS}_{(Z,\mathsf{iter})}$ becomes $\emptyset$. The proof now follows from the fact that in each hop, there are $\mathcal{O}(n)$ calls to $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$. □

Claim 3.13. In protocol Π_{OptMult}, if no party in $\mathcal{P} \setminus Z$ behaves maliciously, then for each $P_i \in \mathsf{Selected}_{(Z,\mathsf{iter})}$, the condition $c^{(i)}_{(Z,\mathsf{iter})} = \sum_{(p,q) \in \mathsf{SIS}^{(i)}_{(Z,\mathsf{iter})}} [a]_p[b]_q$ holds and $c_{(Z,\mathsf{iter})} = ab$.

Proof. From the protocol steps, it follows that $\mathsf{Selected}_{(Z,\mathsf{iter})} \cap Z = \emptyset$, as no honest part ever votes for any party from Z as a candidate summand-sharing party during any hop in the protocol. Now since $\mathsf{Selected}_{(Z,\mathsf{iter})} \subseteq (\mathcal{P} \setminus Z)$, if no party in $\mathcal{P} \setminus Z$ behaves maliciously, then it implies that every party $P_i \in \mathsf{Selected}_{(Z,\mathsf{iter})}$ behaves honestly and secret-shares $c^{(i)}_{(Z,\mathsf{iter})}$ by calling $\mathcal{F}_{\mathsf{VSS}}$, where

$$c^{(i)}_{(Z,\mathsf{iter})} = \sum_{(p,q) \in \mathsf{SIS}^{(i)}_{(Z,\mathsf{iter})}} [a]_p[b]_q.$$

Moreover, from the protocol steps, it follows that for every $P_j, P_k \in \mathsf{Selected}_{(Z,\mathsf{iter})}$:

$$\mathsf{SIS}^{(j)}_{(Z,\mathsf{iter})} \cap \mathsf{SIS}^{(k)}_{(Z,\mathsf{iter})} = \emptyset.$$

To prove this, suppose P_j and P_k are included in $\mathsf{Selected}_{(Z,\mathsf{iter})}$ during hop number hop_j and hop_k respectively, where without loss of generality, $\mathsf{hop}_j < \mathsf{hop}_k$. Then from the protocol steps, during hop_j, the parties would set $\mathsf{SIS}^{(k)}_{(Z,\mathsf{iter})} = \mathsf{SIS}^{(k)}_{(Z,\mathsf{iter})} \setminus \mathsf{SIS}^{(j)}_{(Z,\mathsf{iter})}$. This ensures that during hop_k, there exists no ordered pair $(p, q) \in \{1, \ldots, |\mathbb{S}|\} \times \{1, \ldots, |\mathbb{S}|\}$, such that $(p, q) \in \mathsf{SIS}^{(j)}_{(Z,\mathsf{iter})} \cap \mathsf{SIS}^{(k)}_{(Z,\mathsf{iter})}$.

If all the parties $P_i \in \mathsf{Selected}_{(Z,\mathsf{iter})}$ behave honestly, then from the protocol steps, it also follows that :

$$\bigcup_{P_i \in \mathsf{Selected}_{(Z,\mathsf{iter})}} \mathsf{SIS}^{(i)}_{(Z,\mathsf{iter})} = \{(p, q)\}_{p,q=1,\ldots,|\mathbb{S}|}.$$

Finally, from the protocol steps, it follows that $\forall P_j \in \mathcal{P} \setminus \mathsf{Selected}_{(Z,\mathsf{iter})}$, the condition $c^{(j)}_{(Z,\mathsf{iter})} = 0$ holds. Now since $c_{(Z,\mathsf{iter})} = c^{(1)}_{(Z,\mathsf{iter})} + \ldots + c^{(n)}_{(Z,\mathsf{iter})}$, it follows that if no party in $\mathcal{P} \setminus Z$ behaves maliciously, then $c_{(Z,\mathsf{iter})} = ab$ holds. □

Claim 3.14. In Π_{OptMult}, no additional information about a and b is leaked to Adv.

Proof. Let $Z^\star \in \mathcal{Z}$ be the set of corrupt parties. To prove the claim, we argue that in the protocol, Adv does not learn any additional information about the shares $\{[a]_p, [b]_p\}_{S_p \cap Z^\star = \emptyset}$. For this, consider an arbitrary summand $[a]_p[b]_q$ where $S_p \cap Z^\star = \emptyset$ and where $q \in \{1, \ldots, |\mathbb{S}|\}$. Clearly, the summand $[a]_p[b]_q$ will not be available with any party in $Z^\star$. Let P_j be the party from $\mathsf{Selected}_{(Z,\mathsf{iter})}$, such that $(p, q) \in \mathsf{SIS}^{(j)}_{(Z,\mathsf{iter})}$; i.e. the summand $[a]_p[b]_q$ is included by P_j while computing the summand-sum $c^{(j)}_{(Z,\mathsf{iter})}$. Clearly P_j is *honest*, since $P_j \notin Z^\star$. In the protocol, party P_j randomly secret-shares the summand-sum $c^{(j)}_{(Z,\mathsf{iter})}$ by supplying a random vector of shares for $c^{(j)}_{(Z,\mathsf{iter})}$ to the corresponding $\mathcal{F}_{\mathsf{VSS}}$. Now, since $\mathbb{S}$ is $\mathcal{Z}$-*private*, it follows that the shares $\{[c^{(j)}_{(Z,\mathsf{iter})}]_r\}_{S_r \cap Z^\star \neq \emptyset}$ learnt by Adv in the protocol will be independent of the summand $[a]_p[b]_q$ and hence, independent of $[a]_p$. Using a similar argument, we can conclude that the shares learnt by Adv in the protocol will be independent of the summands $[a]_q[b]_p$ (and hence independent of $[b]_p$), where $S_p \cap Z^\star = \emptyset$, and where $q \in \{1, \ldots, |\mathbb{S}|\}$. □

The proof of Lemma 3.15 now follows from Claims 3.10-3.14.

Lemma 3.15. *Let $\mathcal{Z}$ satisfy the $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition and let $\mathbb{S} = \{\mathcal{P} \setminus Z | Z \in \mathcal{Z}\}$. Consider an arbitrary $Z \in \mathcal{Z}$ and* iter*, such that all honest parties participate in the instance $\Pi_{\mathsf{OptMult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], Z, \mathsf{iter})$. Then all honest parties eventually compute $[c_{(Z,\mathsf{iter})}], [c^{(1)}_{(Z,\mathsf{iter})}], \ldots, [c^{(n)}_{(Z,\mathsf{iter})}]$ where $c_{(Z,\mathsf{iter})} = c^{(1)}_{(Z,\mathsf{iter})} + \ldots + c^{(n)}_{(Z,\mathsf{iter})}$, provided no honest party is included in the $\mathcal{GD}$ and $\mathcal{LD}^{(i)}_{\mathsf{iter}'}$ sets and each honest party in the $\mathcal{W}^{(i)}_{\mathsf{iter}'}$ sets of every honest P_i is eventually removed, for all $\mathsf{iter}' < \mathsf{iter}$. If no party in $\mathcal{P} \setminus Z$ acts maliciously, then $c_{(Z,\mathsf{iter})} = ab$. In the protocol,* Adv *does not learn anything additional about a and b. The protocol makes $\mathcal{O}(n^2)$ calls to $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$.*

We end this section by claiming an important property about the protocol Π_{OptMult}. This will be useful later when we analyze the properties of the protocol Π_{Mult}, where Π_{OptMult} is used as a sub-protocol.

Claim 3.16. For every $Z \in \mathcal{Z}$ and every iter, all the following hold for every $P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}$ during the instance $\Pi_{\mathsf{OptMult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], Z, \mathsf{iter})$.

- There exists at least one honest party P_i such that P_j will not be present in the $\mathcal{W}^{(i)}_{\mathsf{iter}'}$ and $\mathcal{LD}^{(i)}_{\mathsf{iter}'}$ sets of P_i for any $\mathsf{iter}' < \mathsf{iter}$.
- P_j will not be present in the set $\mathcal{GD}$.

Proof. Consider an arbitrary $P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}$, such that P_j is included in $\mathsf{Selected}_{(Z,\mathsf{iter})}$ during the hop number hop in the instance $\Pi_{\mathsf{OptMult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], Z, \mathsf{iter})$. We prove the first part of the claim through a contradiction. Let $\mathcal{H}$ be the set of *honest* parties and for every $P_i \in \mathcal{H}$, let there exist some $\mathsf{iter}' < \mathsf{iter}$, such that either $P_j \in \mathcal{W}^{(i)}_{\mathsf{iter}'}$ or $P_j \in \mathcal{LD}^{(i)}_{\mathsf{iter}'}$. This implies that during hop number hop, no $P_i \in \mathcal{H}$ will send $(\mathsf{vote}, \mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}, 1)$ to $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}$. Consequently, $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}$ will never return the output $(\mathsf{decide}, \mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}, 1)$ for any honest party and hence, P_j will not be selected as the summand-sharing party for hop number hop, which is a contradiction.

The second part of the claim also follows using a similar argument as above. Namely, if P_j is present in the set $\mathcal{GD}$, then no $P_i \in \mathcal{H}$ will send $(\mathsf{vote}, \mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}, 1)$ to $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{\mathsf{hop},j,\mathsf{iter},Z}$ and consequently, P_j will not be selected as the summand-sharing party for hop number hop, which is a contradiction. □

Protocol Π_{OptMult} for M Pairs of Inputs. Protocol Π_{OptMult} can be easily modified if there are M pairs of inputs. In each hop, every party will now be sharing M number of summations, by calling $\mathcal{F}_{\mathsf{VSS}}$ M times. Thus, the total number of calls to $\mathcal{F}_{\mathsf{VSS}}$ will be $\mathcal{O}(n^2 M)$. While voting for a candidate summand-sharing party in a hop, the parties check whether it has shared M values. In this way, the number of calls to $\mathcal{F}_{\mathsf{ABA}}$ can be restricted to *only* $\mathcal{O}(n^2)$, *independent* of M.

3.2.2 Multiplication Protocol with Cheater Identification

Based on protocol Π_{OptMult}, we next present the protocol Π_{MultCI} with cheater identification (Fig 11). The protocol takes as inputs an iteration number iter and $[a], [b]$. If *no* party behaves maliciously, then the protocol outputs $[ab]$. In the protocol, parties execute an instance of Π_{OptMult} for each $Z \in \mathcal{Z}$ and *publicly* compare the outputs. Since at least one of the Π_{OptMult} instances is guaranteed to output $[ab]$, if all the outputs are the same, then it implies that no cheating has occurred. Otherwise, the parties identify a pair of *conflicting* Π_{OptMult} instances with different outputs, executed with respect to the sets, say $Z, Z' \in \mathcal{Z}$. Note that this process of publicly comparing the outputs of the Π_{OptMult} instances and identifying a pair of conflicting instances does not reveal any additional information about a and b to the adversary, apart from the differences between various outputs, which will be known beforehand to the adversary.

Let $\mathsf{Selected}_{(Z,\mathsf{iter})}$ and $\mathsf{Selected}_{(Z',\mathsf{iter})}$ be the summand-sharing parties in the conflicting Π_{OptMult} instances. The parties next proceed to a cheater-identification phase to identify at least one corrupt party from the set $\mathsf{Selected}_{(Z,\mathsf{iter})} \cup \mathsf{Selected}_{(Z',\mathsf{iter})}$. For this, each summand-sharing party $P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}$ is made to re-share the sum of the summands, overlapping with the summands whose sum has been secret-shared by summand-sharing parties $P_k \in \mathsf{Selected}_{(Z',\mathsf{iter})}$ and vice-versa. The re-shared secret-shared sums are "compared" with the summations secret-shared earlier by the summand-sharing parties during the instances of Π_{OptMult}. This is to prevent a corrupt summand-sharing party from re-sharing different sums than what had been secret-shared earlier. Next, these "partitions" are compared, based on which at least one corrupt party in $\mathsf{Selected}_{(Z,\mathsf{iter})} \cup \mathsf{Selected}_{(Z',\mathsf{iter})}$ is guaranteed to be identified provided *all* the parties in $\mathsf{Selected}_{(Z,\mathsf{iter})} \cup \mathsf{Selected}_{(Z',\mathsf{iter})}$ secret-share the required partitions.

The cheater-identification phase will be "stuck" if the *corrupt* parties in $\mathsf{Selected}_{(Z,\mathsf{iter})} \cup \mathsf{Selected}_{(Z',\mathsf{iter})}$ do not participate and secret-share the required partitions. To prevent such corrupt parties from causing future instances of Π_{MultCI} to fail, the parties wait-list all the parties in $\mathsf{Selected}_{(Z,\mathsf{iter})} \cup \mathsf{Selected}_{(Z',\mathsf{iter})}$. A party is then "released" only after it has re-shared all the required values as part of the cheater-identification phase. Notice that every honest party from $\mathsf{Selected}_{(Z,\mathsf{iter})} \cup \mathsf{Selected}_{(Z',\mathsf{iter})}$ is eventually released from the waiting-list of every honest party. This wait-listing guarantees that corrupt parties will be barred from acting as summand-sharing parties as part of the Π_{OptMult} instances of future invocations of Π_{MultCI}, until they participate in the cheater-identification phase of previously failed instances of Π_{MultCI}. Since the cheater-identification phase is executed asynchronously, each party maintains its own set of *locally-discarded* parties, where corrupt parties are included as and when they are identified.

Protocol $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], \mathsf{iter})$

- **Initialization**: Initialize $\mathcal{W}^{(i)}_{\mathsf{iter}} = \mathcal{LD}^{(i)}_{\mathsf{iter}} = \emptyset$ and $\mathsf{flag}^{(i)}_{\mathsf{iter}} = \perp$.[a] Fix some (publicly-known) $Z' \in \mathcal{Z}$.
- **Running Optimistic Multiplication and Checking Pair-wise Differences**:
  - For each $Z \in \mathcal{Z}$, participate in the instance $\Pi_{\mathsf{OptMult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], Z, \mathsf{iter})$ with session id sid. Let $\{[c^{(1)}_{(Z,\mathsf{iter})}]_q, \ldots, [c^{(n)}_{(Z,\mathsf{iter})}]_q, [c_{(Z,\mathsf{iter})}]_q\}_{P_i \in S_q}$ be the output obtained. Moreover, let $\mathsf{Selected}_{(Z,\mathsf{iter})}$ be set of summand-sharing parties and for each $P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}$, let $\mathsf{SIS}^{(j)}_{(Z,\mathsf{iter})}$ be the summand-index-set of ordered pairs of indices corresponding to the summands whose sum has been secret-shared by P_j, during this instance of Π_{OptMult}.
  - Corresponding to every $Z \in \mathcal{Z}$ where $Z \neq Z'$, participate in an instance of Π_{PerRec} to publicly reconstruct $c_{(Z,\mathsf{iter})} - c_{(Z',\mathsf{iter})}$.
- **Output in Case of Success**: If $c_{(Z,\mathsf{iter})} - c_{(Z',\mathsf{iter})} = 0$ for every $Z \in \mathcal{Z}$, then do the following.
  - Set $\mathsf{flag}^{(i)}_{\mathsf{iter}} = 0$;
  - Output $\{[c_{(Z',\mathsf{iter})}]_q\}_{P_i \in S_q}$.
- **Waiting-List and Cheater Identification in Case of Failure**: If there exists a $Z \in \mathcal{Z}$ such that $c_{(Z,\mathsf{iter})} - c_{(Z',\mathsf{iter})} \neq 0$, then let Z be the first set from $\mathcal{Z}$ such that $c_{(Z,\mathsf{iter})} - c_{(Z',\mathsf{iter})} \neq 0$. Set the *conflicting-sets* to be Z, Z', set $\mathsf{flag}^{(i)}_{\mathsf{iter}} = 1$ and proceed as follows.
  - **Wait-listing Parties**: Set $\mathcal{W}^{(i)}_{\mathsf{iter}} = \mathsf{Selected}_{(Z,\mathsf{iter})} \cup \mathsf{Selected}_{(Z',\mathsf{iter})}$.
  - **Re-Sharing Partition of the Summand-Sums**:
    1. If $P_i \in \mathsf{Selected}_{(Z,\mathsf{iter})}$, then for every $P_j \in \mathsf{Selected}_{(Z',\mathsf{iter})}$, compute:
    $$d^{(ij)}_{(Z,\mathsf{iter})} = \sum_{(p,q) \in \mathsf{SIS}^{(i)}_{(Z,\mathsf{iter})} \cap \mathsf{SIS}^{(j)}_{(Z',\mathsf{iter})}} [a]_p [b]_q.$$
    Randomly pick $d^{(ij)}_{(Z,\mathsf{iter})_1}, \ldots, d^{(ij)}_{(Z,\mathsf{iter})_h}$ such that $d^{(ij)}_{(Z,\mathsf{iter})_1} + \ldots + d^{(ij)}_{(Z,\mathsf{iter})_h} = d^{(ij)}_{(Z,\mathsf{iter})}$. Send $(\mathsf{dealer}, \mathsf{sid}_{i,j,\mathsf{iter},Z}, (d^{(ij)}_{(Z,\mathsf{iter})_1}, \ldots, d^{(ij)}_{(Z,\mathsf{iter})_h})$ to $\mathcal{F}_{\mathsf{VSS}}$, where $\mathsf{sid}_{i,j,\mathsf{iter},Z} = \mathsf{sid}||i||j||\mathsf{iter}||Z$.
    2. If $P_i \in \mathsf{Selected}_{(Z',\mathsf{iter})}$, then for all $P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}$, compute:
    $$e^{(ij)}_{(Z',\mathsf{iter})} = \sum_{(p,q) \in \mathsf{SIS}^{(i)}_{(Z',\mathsf{iter})} \cap \mathsf{SIS}^{(j)}_{(Z,\mathsf{iter})}} [a]_p [b]_q.$$
    Randomly pick $e^{(ij)}_{(Z',\mathsf{iter})_1}, \ldots, e^{(ij)}_{(Z',\mathsf{iter})_h}$ such that $e^{(ij)}_{(Z',\mathsf{iter})_1} + \ldots + e^{(ij)}_{(Z',\mathsf{iter})_h} = e^{(ij)}_{(Z',\mathsf{iter})}$. Send $(\mathsf{dealer}, \mathsf{sid}_{i,j,\mathsf{iter},Z'}, (e^{(ij)}_{(Z',\mathsf{iter})_1}, \ldots, e^{(ij)}_{(Z',\mathsf{iter})_h})$ to $\mathcal{F}_{\mathsf{VSS}}$, where $\mathsf{sid}_{i,j,\mathsf{iter},Z'} = \mathsf{sid}||i||j||\mathsf{iter}||Z'$.
    3. Corresponding to every $P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}$ and every $P_k \in \mathsf{Selected}_{(Z',\mathsf{iter})}$, keep requesting for an output from $\mathcal{F}_{\mathsf{VSS}}$ with session id $\mathsf{sid}_{j,k,\mathsf{iter},Z}$, till an output is obtained.
    4. Corresponding to every $P_j \in \mathsf{Selected}_{(Z',\mathsf{iter})}$ and every $P_k \in \mathsf{Selected}_{(Z,\mathsf{iter})}$, keep requesting for an output from $\mathcal{F}_{\mathsf{VSS}}$ with session id $\mathsf{sid}_{j,k,\mathsf{iter},Z'}$, till an output is obtained.
  - **Removing Parties from Wait List**: Set $\mathcal{W}^{(i)}_{\mathsf{iter}} = \mathcal{W}^{(i)}_{\mathsf{iter}} \setminus \{P_j\}$, if *all* the following criteria pertaining to P_j hold:
    1. $P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}$: if an output $(\mathsf{share}, \mathsf{sid}_{j,k,\mathsf{iter},Z}, P_j, \{[d^{(jk)}_{(Z,\mathsf{iter})}]_q\}_{P_i \in S_q})$ is received from $\mathcal{F}_{\mathsf{VSS}}$ with session id $\mathsf{sid}_{j,k,\mathsf{iter},Z}$, corresponding to each $P_k \in \mathsf{Selected}_{(Z',\mathsf{iter})}$,
    2. $P_j \in \mathsf{Selected}_{(Z',\mathsf{iter})}$: if an output $(\mathsf{share}, \mathsf{sid}_{j,k,\mathsf{iter},Z'}, P_j, \{[e^{(jk)}_{(Z',\mathsf{iter})}]_q\}_{P_i \in S_q})$ is received from $\mathcal{F}_{\mathsf{VSS}}$ with session id $\mathsf{sid}_{j,k,\mathsf{iter},Z'}$, corresponding to every $P_k \in \mathsf{Selected}_{(Z,\mathsf{iter})}$.
  - **Verifying the Re-Shared Summand-Sum Partitions and Locally Identifying Corrupt Parties**:
    1. For every $P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}$, participate in an instance of Π_{PerRec} to reconstruct the following difference value:
    $$c^{(j)}_{(Z,\mathsf{iter})} - \sum_{P_k \in \mathsf{Selected}_{(Z',\mathsf{iter})}} d^{(jk)}_{(Z,\mathsf{iter})}.$$

If the difference is not 0, then set $\mathcal{LD}^{(i)}_{\mathsf{iter}} = \mathcal{LD}^{(i)}_{\mathsf{iter}} \cup \{P_j\}$.

2. For every $P_j \in \mathsf{Selected}_{(Z',\mathsf{iter})}$, participate in an instance of Π_{PerRec} to reconstruct the following difference value:

$$c^{(j)}_{(Z',\mathsf{iter})} - \sum_{P_k \in \mathsf{Selected}_{(Z,\mathsf{iter})}} e^{(jk)}_{(Z',\mathsf{iter})}.$$

If the difference is not 0, then set $\mathcal{LD}^{(i)}_{\mathsf{iter}} = \mathcal{LD}^{(i)}_{\mathsf{iter}} \cup \{P_j\}$.

3. For each ordered pair (P_j, P_k) where $P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}$ and $P_k \in \mathsf{Selected}_{(Z',\mathsf{iter})}$, participate in an instance of Π_{PerRec} to reconstruct the following difference value:

$$d^{(jk)}_{(Z,\mathsf{iter})} - e^{(kj)}_{(Z',\mathsf{iter})}.$$

If the difference is not 0, then do the following:

i. Participate in instances of Π_{PerRec} to reconstruct $d^{(jk)}_{(Z,\mathsf{iter})}$ and $e^{(kj)}_{(Z',\mathsf{iter})}$.

ii. Participate in instances of $\Pi_{\mathsf{PerRecShare}}$ to reconstruct the shares $[a]_p$ and $[b]_q$, such that $(p, q) \in \mathsf{SIS}^{(j)}_{(Z,\mathsf{iter})} \cap \mathsf{SIS}^{(k)}_{(Z',\mathsf{iter})}$.

iii. Compute the sum $f^{(jk)}_{(\mathsf{iter})}$, where:

$$f^{(jk)}_{(\mathsf{iter})} = \sum_{(p,q) \in \mathsf{SIS}^{(j)}_{(Z,\mathsf{iter})} \cap \mathsf{SIS}^{(k)}_{(Z',\mathsf{iter})}} [a]_p [b]_q.$$

iv. If $f^{(jk)}_{(\mathsf{iter})} \neq d^{(jk)}_{(Z,\mathsf{iter})}$, then set $\mathcal{LD}^{(i)}_{\mathsf{iter}} = \mathcal{LD}^{(i)}_{\mathsf{iter}} \cup \{P_j\}$.

v. If $f^{(jk)}_{(\mathsf{iter})} \neq e^{(kj)}_{(Z',\mathsf{iter})}$, then set $\mathcal{LD}^{(i)}_{\mathsf{iter}} = \mathcal{LD}^{(i)}_{\mathsf{iter}} \cup \{P_k\}$.

[a] The variable $\mathsf{flag}^{(i)}_{\mathsf{iter}}$ will be used later as an indicator if any cheating is detected during the instance iter of Π_{MultCI}.

Figure 11: Code for P_i for multiplication with cheater identification for iteration iter and session id sid, in the $\mathcal{F}_{\mathsf{VSS}}$-hybrid

For a better understanding of the various sets and values computed during the protocol Π_{MultCI}, we pictorially illustrate them in Fig 12.

We next proceed to prove the properties of the protocol Π_{MultCI}. While proving these properties, we will assume that $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition. This further implies that the sharing specification $\mathbb{S} = \{S_1, \ldots, S_h\} \stackrel{def}{=} \{\mathcal{P} \setminus Z | Z \in \mathcal{Z}\}$ satisfies the $\mathbb{Q}^{(3)}(\mathbb{S}, \mathcal{Z})$ condition. Moreover, we will also assume that *no* honest party is ever included in the set $\mathcal{GD}$, which will be guaranteed in the protocol Π_{Mult} where the set $\mathcal{GD}$ is constructed and managed, and where Π_{MultCI} is used as a sub-protocol.

We first give the definition of a *successful* Π_{MultCI} instance, which will be used throughout this section and the next.

Definition 3.17 (Successful Π_{MultCI} Instance). For an instance $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], \mathsf{iter})$, we define the following.

– The instance is called *successful* if and only if for every $Z \in \mathcal{Z}$ where $Z \neq Z'$, the value $c_{(Z,\mathsf{iter})} - c_{(Z',\mathsf{iter})} = 0$, where $Z' \in \mathcal{Z}$ is the fixed set used in the protocol.

– If the instance is not successful, then the sets Z, Z' are called the *conflicting-sets* for the instance, if Z is the smallest indexed set from $\mathcal{Z}$ such that $c_{(Z,\mathsf{iter})} - c_{(Z',\mathsf{iter})} \neq 0$.

We first show that any instance of Π_{MultCI} will be eventually found to be either a success or a failure by the honest parties.

Claim 3.18. For every iter, any instance $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], \mathsf{iter})$ will eventually be deemed to either succeed or fail by the honest parties, provided no honest party is ever included in the $\mathcal{GD}$ and $\mathcal{LD}^{(i)}_{\mathsf{iter}'}$ sets,

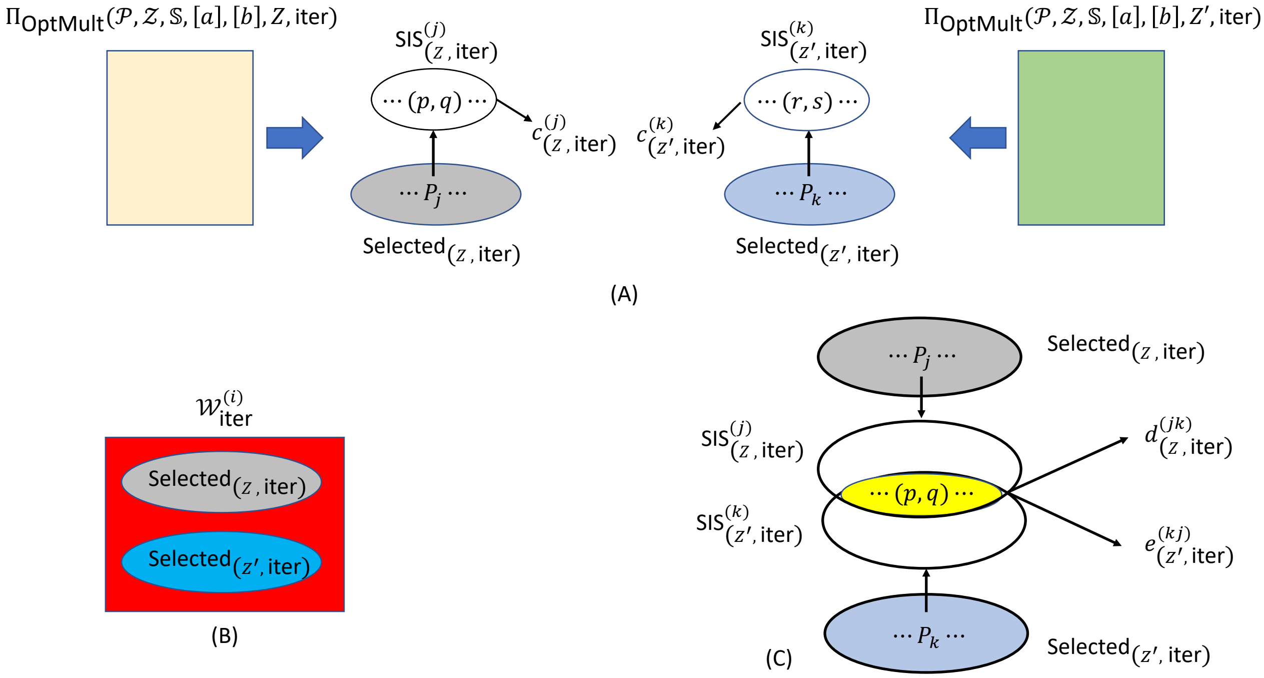

Figure 12: Pictorial depiction of the various values and sets computed during the protocol Π_{MultCI}. Figure (A) shows the pair of conflicting Π_{OptMult} instances corresponding to sets $Z, Z' \in \mathcal{Z}$, with different secret-shared outputs $c_{(Z,\mathsf{iter})}$ and $c_{(Z',\mathsf{iter})}$. The figure also shows the set of summand-sharing parties $\mathsf{Selected}_{(Z,\mathsf{iter})}$ and $\mathsf{Selected}_{(Z',\mathsf{iter})}$ during the two Π_{OptMult} instances. Moreover, for every summand-sharing party $P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}$, there is a summand-index-set $\mathsf{SIS}^{(j)}_{(Z,\mathsf{iter})}$, denoting all the ordered pair of indices whose corresponding summands have been summed to $c^{(j)}_{(Z,\mathsf{iter})}$ and secret-shared by P_j during the corresponding Π_{OptMult} instance. Figure (B) denotes the waiting-list maintained by party P_i, which has all the summand-sharing parties from the conflicting Π_{OptMult} instances. Figure (C) denotes the common summands (whose indices are highlighted in yellow), held by both $P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}$ and $P_k \in \mathsf{Selected}_{(Z',\mathsf{iter})}$, whose corresponding sums $d^{(jk)}_{(Z,\mathsf{iter})}$ and $e^{(kj)}_{(Z',\mathsf{iter})}$ are re-shared by P_j and P_k respectively, during the cheater-identification phase.

and all honest parties are eventually removed from the $\mathcal{W}^{(i)}_{\mathsf{iter}'}$ sets of every honest P_i for every $\mathsf{iter}' < \mathsf{iter}$. Moreover, for a successful instance, the parties output a sharing of ab. If the instance is not successful, then the parties identify the *conflicting-sets* Z, Z' for the instance.

Proof. Let $Z^\star \in \mathcal{Z}$ be the set of corrupt parties. If the lemma conditions hold, then it follows from Lemma 3.15, that corresponding to every $Z \in \mathcal{Z}$, the instance $\Pi_{\mathsf{OptMult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], Z, \mathsf{iter})$ eventually completes with honest parties obtaining the outputs $[c^{(1)}_{(Z,\mathsf{iter})}], \ldots, [c^{(n)}_{(Z,\mathsf{iter})}], [c_{(Z,\mathsf{iter})}]$, where $c_{(Z,\mathsf{iter})} = c^{(1)}_{(Z,\mathsf{iter})} + \ldots + c^{(n)}_{(Z,\mathsf{iter})}$. Moreover, in the Π_{OptMult} instance corresponding to $Z^\star$, the output $c_{(Z^\star,\mathsf{iter})}$ will be the same as ab, since all the parties in $\mathcal{P} \setminus Z^\star$ will be honest.

Since $\mathbb{S}$ satisfies the $\mathbb{Q}^{(3)}(\mathbb{S}, \mathcal{Z})$ condition, it follows that with respect to the fixed $Z' \in \mathcal{Z}$, the honest parties will eventually reconstruct the difference $c_{(Z,\mathsf{iter})} - c_{(Z',\mathsf{iter})}$, corresponding to *every* $Z \in \mathcal{Z}$. Now there are two possibilities. If all the differences $c_{(Z,\mathsf{iter})} - c_{(Z',\mathsf{iter})}$ turn out to be 0, then the Π_{MultCI} instance will be considered to be successful by the honest parties and the honest parties will output $[c_{(Z',\mathsf{iter})}]$, which is bound to be the same as ab. This is because $c_{(Z',\mathsf{iter})} - c_{(Z^\star,\mathsf{iter})} = 0$ and hence $c_{(Z',\mathsf{iter})} = c_{(Z^\star,\mathsf{iter})} = ab$ holds. The other possibility is that *at least* one of the differences is not zero, in which case the instance Π_{MultCI} will not be considered successful by the honest parties. Moreover, in this case, the parties will set (Z, Z') as the conflicting-sets for the instance, where Z is the smallest indexed set from $\mathcal{Z}$ such that $c_{(Z,\mathsf{iter})} - c_{(Z',\mathsf{iter})} \neq 0$. □

We next prove a series of claims regarding any Π_{MultCI} instance which is *not* successful. We begin by showing that if any instance of Π_{MultCI} is *not* successful, then every honest party in eventually removed from the waiting-list of the honest parties for that instance. Moreover, no honest party will be ever included in the $\mathcal{LD}$ set of any honest party for that instance.

Claim 3.19. For every iter, if the instance $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], \mathsf{iter})$ is not successful, then every honest party from the set $\mathsf{Selected}_{(Z,\mathsf{iter})} \cup \mathsf{Selected}_{(Z',\mathsf{iter})}$ is eventually removed from the waiting set $\mathcal{W}^{(i)}_{\mathsf{iter}}$ of every honest party P_i. Moreover, no honest party is ever included in the $\mathcal{LD}^{(i)}_{\mathsf{iter}}$ set of any honest party P_i.

Proof. Suppose that the instance $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], \mathsf{iter})$ is not successful. This implies that the parties identify a pair of conflicting-sets (Z, Z'), such that $c_{(Z,\mathsf{iter})} - c_{(Z',\mathsf{iter})} \neq 0$. From the protocol steps, every honest party P_i initializes $\mathcal{W}^{(i)}_{\mathsf{iter}}$ to $\mathsf{Selected}_{(Z,\mathsf{iter})} \cup \mathsf{Selected}_{(Z',\mathsf{iter})}$. Let P_j be an arbitrary *honest* party belonging to $\mathsf{Selected}_{(Z,\mathsf{iter})} \cup \mathsf{Selected}_{(Z',\mathsf{iter})}$. From the protocol steps, party P_j secret-shares all the required values $d^{(jk)}_{(Z,\mathsf{iter})}, e^{(jk)}_{(Z',\mathsf{iter})}$ by calling appropriate instances of $\mathcal{F}_{\mathsf{VSS}}$ and eventually, these values are secret-shared, with every honest P_i receiving the appropriate shares from corresponding $\mathcal{F}_{\mathsf{VSS}}$ instances. Consequently, P_j will eventually be removed from the set $\mathcal{W}^{(i)}_{\mathsf{iter}}$. Moreover, since P_j is an *honest* party, the $d^{(jk)}_{(Z,\mathsf{iter})}, e^{(jk)}_{(Z',\mathsf{iter})}$ values shared by P_j will be correct and consequently, the conditions for including P_j to the $\mathcal{LD}^{(i)}_{\mathsf{iter}}$ set of any honest party P_i will fail. That is, if $P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}$, then the parties will find that the following holds:

$$c^{(j)}_{(Z,\mathsf{iter})} - \sum_{P_k \in \mathsf{Selected}_{(Z',\mathsf{iter})}} d^{(jk)}_{(Z,\mathsf{iter})} = 0.$$

On the other hand, if $P_j \in \mathsf{Selected}_{(Z',\mathsf{iter})}$, then the parties will find that the following holds:

$$c^{(j)}_{(Z',\mathsf{iter})} - \sum_{P_k \in \mathsf{Selected}_{(Z,\mathsf{iter})}} e^{(jk)}_{(Z',\mathsf{iter})} = 0.$$

Moreover, if there exists any $P_k \in \mathsf{Selected}_{(Z,\mathsf{iter})} \cup \mathsf{Selected}_{(Z',\mathsf{iter})}$ such that either $d^{(jk)}_{(Z,\mathsf{iter})} \neq e^{(kj)}_{(Z',\mathsf{iter})}$ or $d^{(kj)}_{(Z,\mathsf{iter})} \neq e^{(jk)}_{(Z',\mathsf{iter})}$, then after reconstructing the values shared by P_j and the shares held by P_j, the parties will find that P_j has behaved honestly and hence, P_j will not be included to the $\mathcal{LD}^{(i)}_{\mathsf{iter}}$ set of any honest P_i. □

We next give the definition of a *conflicting-pair* of parties, which is defined based on the partitions of the summand-sum, re-shared by the summand-sharing parties.

Definition 3.20 (Conflicting-Pair of Parties). Let $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], \mathsf{iter})$ be an instance of Π_{MultCI} which is not successful and let Z, Z' be the corresponding conflicting-sets for the instance. A pair of parties (P_j, P_k) is said to be a *conflicting-pair* of parties for this Π_{MultCI} instance if *all* the following hold:

– $P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}$;
– $P_k \in \mathsf{Selected}_{(Z',\mathsf{iter})}$;
– $d^{(jk)}_{(Z,\mathsf{iter})} \neq e^{(kj)}_{(Z',\mathsf{iter})}$.

We next show that if an instance of Π_{MultCI} is not successful, then certain conditions hold with respect to the summand-sums and the respective partitions shared by the summand-sharing parties during the underlying instances of Π_{OptMult} and the cheater-identification phase of the Π_{MultCI} instance.

Claim 3.21. Let $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], \mathsf{iter})$ be an instance of Π_{MultCI} which is not successful and let Z, Z' be the corresponding conflicting-sets for the instance. Moreover, let $Z^\star$ be the set of corrupt parties. Then, one of the following must hold true for some $P_j \in Z^\star$.

i. $P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}$ and $c^{(j)}_{(Z,\mathsf{iter})} - \sum_{P_k \in \mathsf{Selected}_{(Z',\mathsf{iter})}} d^{(jk)}_{(Z,\mathsf{iter})} \neq 0$.

ii. $P_j \in \mathsf{Selected}_{(Z',\mathsf{iter})}$ and $c^{(j)}_{(Z',\mathsf{iter})} - \sum_{P_k \in \mathsf{Selected}_{(Z,\mathsf{iter})}} e^{(jk)}_{(Z',\mathsf{iter})} \neq 0$.

iii. There is some $P_k \in \mathsf{Selected}_{(Z,\mathsf{iter})} \cup \mathsf{Selected}_{(Z',\mathsf{iter})}$ such that either (P_j, P_k) or (P_k, P_j) constitutes a conflicting-pair of parties.

Proof. Since the instance of Π_{MultCI} is not successful and Z, Z' constitute conflicting-sets, it follows that $c_{(Z,\mathsf{iter})} \neq c_{(Z',\mathsf{iter})}$. Assume for the sake of contradiction that the none of the conditions in the claim is true. Then, we can infer the following.

$$
\begin{aligned}
c_{(Z,\mathsf{iter})} &= \sum_{P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}} c^{(j)}_{(Z,\mathsf{iter})} \\
&= \sum_{P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}} \sum_{P_k \in \mathsf{Selected}_{(Z',\mathsf{iter})}} d^{(jk)}_{(Z,\mathsf{iter})} \\
&= \sum_{P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}} \sum_{P_k \in \mathsf{Selected}_{(Z',\mathsf{iter})}} e^{(kj)}_{(Z',\mathsf{iter})} \\
&= \sum_{P_k \in \mathsf{Selected}_{(Z',\mathsf{iter})}} \sum_{P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}} e^{(kj)}_{(Z',\mathsf{iter})} \\
&= \sum_{P_k \in \mathsf{Selected}_{(Z',\mathsf{iter})}} c^{(k)}_{(Z',\mathsf{iter})} \\
&= c_{(Z',\mathsf{iter})},
\end{aligned}
$$

where the first equation follows from the definition of $c_{(Z,\mathsf{iter})}$, the second equation holds because, as per our assumption, $c^{(j)}_{(Z,\mathsf{iter})} - \sum_{P_k \in \mathsf{Selected}_{(Z',\mathsf{iter})}} d^{(jk)}_{(Z,\mathsf{iter})} = 0$ for every $P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}$, the third equation holds because, as per our assumption, there is *no* conflicting-pair of parties, the fifth equation holds because as per our assumption $c^{(k)}_{(Z',\mathsf{iter})} - \sum_{P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}} e^{(kj)}_{(Z',\mathsf{iter})} = 0$ for every $P_k \in \mathsf{Selected}_{(Z',\mathsf{iter})}$ and the last equation follows from the definition of $c_{(Z',\mathsf{iter})}$. However, $c_{(Z,\mathsf{iter})} = c_{(Z',\mathsf{iter})}$ is a contradiction. □

We next define a *characteristic function* with respect to the partitions of the summands-sum shared by the summand-sharing parties, to "characterize" instances of Π_{MultCI} which are *not* successful. Looking ahead, this will be helpful to upper bound the number of failed Π_{MultCI} instances in the protocol Π_{Mult}.

Definition 3.22 (Characteristic Function). Let $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], \mathsf{iter})$ be an instance of Π_{MultCI} which is not successful and let Z, Z' be the corresponding conflicting-sets for the instance. Then the *characteristic function* f_{char} for this instance is defined as follows.

– If there is some $P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}$ such that $c^{(j)}_{(Z,\mathsf{iter})} - \sum_{P_k \in \mathsf{Selected}_{(Z',\mathsf{iter})}} d^{(jk)}_{(Z,\mathsf{iter})} \neq 0$, then $f_{\mathsf{char}}(\mathsf{iter}) \stackrel{def}{=} (P_j, P_k)$, where P_k is the smallest-indexed party from $\mathcal{P} \setminus \{P_j\}$.[12]

– Else, if there is some $P_j \in \mathsf{Selected}_{(Z',\mathsf{iter})}$ such that $c^{(j)}_{(Z',\mathsf{iter})} - \sum_{P_k \in \mathsf{Selected}_{(Z,\mathsf{iter})}} e^{(jk)}_{(Z',\mathsf{iter})} \neq 0$, then $f_{\mathsf{char}}(\mathsf{iter}) = (P_k, P_j)$, where P_k is the smallest-indexed party from $\mathcal{P} \setminus \{P_j\}$.[13]

– Else, $f_{\mathsf{char}}(\mathsf{iter}) \stackrel{def}{=} (P_j, P_k)$, where (P_j, P_k) is a conflicting-pair of parties, corresponding to the Π_{MultCI} instance.[14]

Before we proceed, we would like to stress that if f_{char} is defined either with respect to the first or the second condition, then party P_k in the pair (P_j, P_k) serves as a "dummy" party. This is just for notational convenience to ensure uniformity so that f_{char} is *always* a pair of parties irrespective of whether it is defined with respect to the first, second or third condition.

From the definition, it is easy to see that if $f_{\mathsf{char}}(\mathsf{iter}) = (P_j, P_k)$, then at least one party among P_j, P_k is *maliciously-corrupt*. We next claim that the characteristic function is well defined.

Claim 3.23. Let $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], \mathsf{iter})$ be an instance of Π_{MultCI} which is not successful and let Z, Z' be the corresponding conflicting-sets for the instance. Then $f_{\mathsf{char}}(\mathsf{iter})$ is well defined.

Proof. Proof follows from Claim 3.21. □

We next prove an important property by showing that if $f_{\mathsf{char}}(\mathsf{iter}) = (P_j, P_k)$ for some instance of Π_{MultCI} which is not successful, and if both P_j and P_k have been removed from the waiting-list of some honest party for that instance, then the corrupt party(ies) among P_j, P_k will eventually be discarded by *all* honest parties.

Claim 3.24. Let $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], \mathsf{iter})$ be an instance of Π_{MultCI} which is not successful and let Z, Z' be the corresponding conflicting-sets for the instance. Moreover, let $f_{\mathsf{char}}(\mathsf{iter}) = (P_j, P_k)$. If both P_j and P_k are removed from the set $\mathcal{W}^{(h)}_{\mathsf{iter}}$ of *any* honest party P_h, then the corrupt party(ies) among P_j, P_k will eventually be included in the set $\mathcal{LD}^{(i)}_{\mathsf{iter}}$ of *every* honest P_i.

Proof. Let $f_{\mathsf{char}}(\mathsf{iter}) = (P_j, P_k)$, where without loss of generality, $P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})}$ and $P_k \in \mathsf{Selected}_{(Z',\mathsf{iter})}$. From the definition of characteristic function (Def 3.22), one of the following holds for P_j and P_k:

– (P_j, P_k) *constitutes a conflicting-pair*: In this case, $d^{(jk)}_{(Z,\mathsf{iter})} \neq e^{(kj)}_{(Z',\mathsf{iter})}$. Since the *honest* P_h has removed both P_j and P_k from $\mathcal{W}^{(h)}_{\mathsf{iter}}$, from the protocol steps, the outputs $(\mathsf{share}, \mathsf{sid}_{j,k,\mathsf{iter},Z}, P_j, \{[d^{(jk)}_{(Z,\mathsf{iter})}]_q\}_{P_h \in S_q})$ and $(\mathsf{share}, \mathsf{sid}_{k,j,\mathsf{iter},Z'}, P_k, \{[e^{(kj)}_{(Z',\mathsf{iter})}]_q\}_{P_h \in S_q})$ have been obtained by P_h from $\mathcal{F}_{\mathsf{VSS}}$ with $\mathsf{sid}_{j,k,\mathsf{iter},Z}$ and $\mathsf{sid}_{k,j,\mathsf{iter},Z'}$ respectively. Consequently, each honest party will eventually receive its respective share corresponding to $[d^{(jk)}_{(Z,\mathsf{iter})}]$ and $[e^{(kj)}_{(Z',\mathsf{iter})}]$ from the corresponding $\mathcal{F}_{\mathsf{VSS}}$ instances. Hence, each honest party will be able to locally compute its share of $d^{(jk)}_{(Z,\mathsf{iter})} - e^{(kj)}_{(Z',\mathsf{iter})}$ and participate in the instance of Π_{PerRec} to reconstruct the difference. Since $\mathbb{S}$ satisfies the $\mathbb{Q}^{(3)}(\mathbb{S}, \mathcal{Z})$ condition, all honest parties will eventually reconstruct $d^{(jk)}_{(Z,\mathsf{iter})} - e^{(kj)}_{(Z',\mathsf{iter})}$ and find that the difference

[12]If there are multiple parties P_j satisfying this condition, then we consider the P_j with the smallest index.
[13]If there are multiple parties P_j satisfying this condition, then we consider the P_j with the smallest index.
[14]If there are multiple conflicting-pairs, then we consider the one having parties with the smallest indices.

is not 0. Consequently, the honest parties will participate in appropriate instances of Π_{PerRec} to reconstruct the values $d^{(jk)}_{(Z,\mathsf{iter})}, e^{(kj)}_{(Z',\mathsf{iter})}$, and instances of $\Pi_{\mathsf{PerRecShare}}$ to reconstruct the shares $[a]_p$ and $[b]_q$, such that $(p,q) \in \mathsf{SIS}^{(j)}_{(Z,\mathsf{iter})} \cap \mathsf{SIS}^{(k)}_{(Z',\mathsf{iter})}$. Now, either $d^{(jk)}_{(Z,\mathsf{iter})} \neq f^{(jk)}_{(\mathsf{iter})}$ or $e^{(kj)}_{(Z',\mathsf{iter})} \neq f^{(jk)}_{(\mathsf{iter})}$, where:

$$f^{(jk)}_{(\mathsf{iter})} = \sum_{(p,q)\in\mathsf{SIS}^{(j)}_{(Z,\mathsf{iter})}\cap\mathsf{SIS}^{(k)}_{(Z',\mathsf{iter})}} [a]_p[b]_q,$$

as otherwise $d^{(jk)}_{(Z,\mathsf{iter})} = e^{(kj)}_{(Z',\mathsf{iter})}$ will hold, which is a contradiction. Consequently, every honest party P_i will eventually add the corrupt party(ies) among P_j, P_k to $\mathcal{LD}^{(i)}_{\mathsf{iter}}$.

– *The condition* $c^{(j)}_{(Z,\mathsf{iter})} - \sum_{P_k\in\mathsf{Selected}_{(Z',\mathsf{iter})}} d^{(jk)}_{(Z,\mathsf{iter})} \neq 0$ *holds*: Since the *honest* P_h has removed P_j from $\mathcal{W}^{(h)}_{\mathsf{iter}}$, then from the protocol steps, corresponding to every $P_k \in \mathsf{Selected}_{(Z',\mathsf{iter})}$, party P_h has received the output $(\mathsf{share}, \mathsf{sid}_{j,k,\mathsf{iter},Z}, P_j, \{[d^{(jk)}_{(Z,\mathsf{iter})}]_q\}_{P_h\in S_q})$ from $\mathcal{F}_{\mathsf{VSS}}$ with $\mathsf{sid}_{j,k,\mathsf{iter},Z}$. Consequently, for every $P_k \in \mathsf{Selected}_{(Z',\mathsf{iter})}$, all honest parties eventually receive their respective shares corresponding to $[d^{(jk)}_{(Z,\mathsf{iter})}]$ from the respective $\mathcal{F}_{\mathsf{VSS}}$ instances. In the protocol, all honest parties participate in an instance of Π_{PerRec} with their respective shares corresponding to $[c^{(j)}_{(Z,\mathsf{iter})}] - \sum_{P_k\in\mathsf{Selected}_{(Z',\mathsf{iter})}} [d^{(jk)}_{(Z,\mathsf{iter})}]$ to reconstruct the difference $c^{(j)}_{(Z,\mathsf{iter})} - \sum_{P_k\in\mathsf{Selected}_{(Z',\mathsf{iter})}} d^{(jk)}_{(Z,\mathsf{iter})}$. Now since the difference is not 0, each honest P_i will eventually include the corrupt P_j to $\mathcal{LD}^{(i)}_{\mathsf{iter}}$.

– *The condition* $c^{(k)}_{(Z',\mathsf{iter})} - \sum_{P_j\in\mathsf{Selected}_{(Z,\mathsf{iter})}} e^{(kj)}_{(Z',\mathsf{iter})} \neq 0$ *holds*: This case is symmetric to the previous case and using a similar argument as above, we can conclude that each honest P_i will eventually include the corrupt P_k to $\mathcal{LD}^{(i)}_{\mathsf{iter}}$.

□

We next claim that the adversary does not learn anything additional about a and b in the protocol.

Claim 3.25. In Π_{MultCI}, Adv does not not learn any additional information about a and b.

Proof. From Claim 3.14, Adv does not learn any additional information about a and b from the instances of Π_{OptMult} executed in Π_{MultCI}. Corresponding to every $Z \in \mathcal{Z}$, Adv learns the difference $c_{(Z,\mathsf{iter})} - c_{(Z',\mathsf{iter})}$ which are all 0, unless the adversary cheats. In case of cheating, the reconstructed differences $c_{(Z,\mathsf{iter})} - c_{(Z',\mathsf{iter})}$ depend completely upon the inputs of the adversary and hence learning these differences does not add anything additional about a and b to the adversary's view. Next, corresponding to every *honest* $P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})} \cup \mathsf{Selected}_{(Z',\mathsf{iter})}$, the shares corresponding to $d^{(jk)}_{(Z,\mathsf{iter})}$ or $e^{(kj)}_{(Z',\mathsf{iter})}$ learnt by Adv will be distributed uniformly, since $\mathbb{S}$ is $\mathcal{Z}$-private and hence, these shares do not add anything additional about a and b to the adversary's view. Moreover, for every *honest* $P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})} \cup \mathsf{Selected}_{(Z',\mathsf{iter})}$, Adv will know beforehand that the differences $c^{(j)}_{(Z,\mathsf{iter})} - \sum_{P_k\in\mathsf{Selected}_{(Z',\mathsf{iter})}} d^{(jk)}_{(Z,\mathsf{iter})}$ as well as $c^{(j)}_{(Z',\mathsf{iter})} - \sum_{P_k\in\mathsf{Selected}_{(Z,\mathsf{iter})}} e^{(jk)}_{(Z',\mathsf{iter})}$ will be 0 and hence, learning these differences does not add anything additional about a and b to adversary's view. On the other hand, for every *corrupt* $P_j \in \mathsf{Selected}_{(Z,\mathsf{iter})} \cup \mathsf{Selected}_{(Z',\mathsf{iter})}$, the above differences completely depend upon the adversary's inputs and hence, reveal no additional information. Finally, if for any ordered pair of parties (P_j, P_k), the condition $d^{(jk)}_{(Z,\mathsf{iter})} \neq e^{(kj)}_{(Z',\mathsf{iter})}$ holds, then at least one among P_j and P_k is *corrupt*. Consequently, the shares $[a]_p$ and

$[b]_q$ where $(p, q) \in \mathsf{SIS}^{(j)}_{(Z,\mathsf{iter})} \cap \mathsf{SIS}^{(k)}_{(Z',\mathsf{iter})}$ reconstructed in this case are already known to the adversary, and do not add anything new to the view of the adversary regarding a and b. □

Claim 3.26. Protocol Π_{MultCI} needs $\mathcal{O}(|\mathcal{Z}| \cdot n^2)$ calls to $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$ and incurs an additional communication of $\mathcal{O}(|\mathcal{Z}|^2 \cdot n^2 \log |\mathbb{F}| + |\mathcal{Z}| \cdot n^4 \log |\mathbb{F}|)$ bits.

Proof. In the protocol, corresponding to each $Z \in \mathcal{Z}$, an instance of Π_{OptMult} is executed. From Lemma 3.15, this will require $\mathcal{O}(|\mathcal{Z}| \cdot n^2)$ calls to $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$. There are $\mathcal{O}(|\mathcal{Z}|)$ instances of Π_{PerRec} to reconstruct $\mathcal{O}(|\mathcal{Z}|)$ difference values for checking whether the instance is successful or not, incurring a communication of $\mathcal{O}(|\mathcal{Z}|^2 \cdot n^2 \log |\mathbb{F}|)$ bits. If the instance is not successful, then there are $\mathcal{O}(n^2)$ calls to $\mathcal{F}_{\mathsf{VSS}}$ to share various summand-sum partitions. To check whether the correct partitions are shared, $\mathcal{O}(n^2)$ values need to be publicly reconstructed through these many instances of Π_{PerRec}, which incurs a communication of $\mathcal{O}(|\mathcal{Z}| \cdot n^4 \log |\mathbb{F}|)$ bits. □

The proof of Lemma 3.27 now follows from Claims 3.18-3.26.

Lemma 3.27. *Let $\mathcal{Z}$ satisfy the $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition and let all honest parties participate in $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], \mathsf{iter})$. Then,* Adv *does not learn any additional information about a and b. Moreover, the following hold.*

- *The instance will eventually be deemed to succeed or fail by the honest parties, where for a successful instance, the parties output a sharing of ab.*
- *If the instance is not successful, then the honest parties will agree on a pair $Z, Z' \in \mathcal{Z}$ such that $c_{(Z,\mathsf{iter})} - c_{(Z',\mathsf{iter})} \neq 0$. Moreover, all honest parties present in the $\mathcal{W}^{(i)}_{\mathsf{iter}}$ set of any honest party P_i will eventually be removed and no honest party is ever included in the $\mathcal{LD}^{(i)}_{\mathsf{iter}}$ set of any honest P_i. Furthermore, there will be a pair of parties P_j, P_k from $\mathsf{Selected}_{(Z,\mathsf{iter})} \cup \mathsf{Selected}_{(Z',\mathsf{iter})}$, with at least one of them being maliciously-corrupt, such that if both P_j and P_k are removed from the set $\mathcal{W}^{(h)}_{\mathsf{iter}}$ of any honest party P_h, then eventually the corrupt party(ies) among P_j, P_k will be included in the set $\mathcal{LD}^{(i)}_{\mathsf{iter}}$ of every honest P_i.*
- *The protocol makes $\mathcal{O}(|\mathcal{Z}| \cdot n^2)$ calls to $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$ and additionally incurs a communication of $\mathcal{O}((|\mathcal{Z}|^2 \cdot n^2 + |\mathcal{Z}| \cdot n^4) \log |\mathbb{F}|)$ bits.*

Protocol Π_{MultCI} for M Pairs of Inputs. The modifications to the protocol Π_{MultCI} for handling M pairs of secret-shared inputs $\{[a^{(\ell)}], [b^{(\ell)}]\}_{\ell=1,\ldots,M}$ are simple. The parties now run instances of Π_{OptMult} handling M pairs of inputs. Corresponding to every pair (Z, Z'), the parties reconstruct M differences. If any of these differences is non-zero, the parties focus on the smallest-indexed $([a^{(\ell)}], [b^{(\ell)}])$ such that $c^{(\ell)}_{(Z,\mathsf{curr})} - c^{(\ell)}_{(Z',\mathsf{curr})} \neq 0$. The parties then proceed to the cheater identification phase with respect to (Z, Z') and $([a^{(\ell)}], [b^{(\ell)}])$. The protocol will require $\mathcal{O}(M \cdot |\mathcal{Z}| \cdot n^2)$ calls to $\mathcal{F}_{\mathsf{VSS}}$, $\mathcal{O}(|\mathcal{Z}| \cdot n^2)$ calls to $\mathcal{F}_{\mathsf{ABA}}$ and additionally incurs a communication of $\mathcal{O}(M \cdot |\mathcal{Z}|^2 \cdot n^2 \log |\mathbb{F}| + |\mathcal{Z}| \cdot n^4 \log |\mathbb{F}|)$ bits.

3.2.3 Robust Multiplication Protocol

Protocol Π_{Mult} (Fig 13) takes inputs $[a], [b]$ and securely generates $[ab]$. The protocol proceeds in iterations, where in each iteration, an instance of Π_{MultCI} is invoked. If the iteration is successful, then the parties take the output of the corresponding Π_{MultCI} instance. Else, they proceed to the next iteration, with the cheater-identification phase of failed Π_{MultCI} instances running in the background. Let t be the cardinality of maximum-sized subset from $\mathcal{Z}$. To upper bound the number of failed iterations, the parties run ACS after every $tn + 1$ failed iterations to "globally" discard a new corrupt party. This is done through calls to $\mathcal{F}_{\mathsf{ABA}}$, where the parties vote for a candidate corrupt party, based on the $\mathcal{LD}$ sets of *all* failed iterations. The idea is that during these $tn+1$ failed iterations, there will be at least one *corrupt* party who is eventually included in the $\mathcal{LD}$ set of *every* honest party. This is because there can be at most tn distinct pairs of "conflicting-parties" across the $tn+1$ failed iterations (follows from Lemma 3.27). At least one conflicting pair, say (P_j, P_k), is guaranteed to repeat among the $tn+1$ failed instances, with *both* P_j and P_k being removed from the previous waiting-lists. Thus, the corrupt party(ies) among P_j, P_k is eventually included to the $\mathcal{LD}$ sets. There can be at most $t(tn+1)$ failed iterations after which *all* the corrupt parties will be discarded and the next iteration is guaranteed to be successful, with only *honest* parties acting as the candidate summand-sharing parties in the underlying instances of Π_{OptMult}.

Protocol $\Pi_{\mathsf{Mult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b])$

- **Initialization**: Set $t = \max\{|Z| : Z \in \mathcal{Z}\}$, initialize $\mathcal{GD} = \emptyset$ and $\mathsf{iter} = 1$.
- **Multiplication with Cheater Identification**: Participate in the instance $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], \mathsf{iter})$ with sid.
  - **Positive Output**: If $\mathsf{flag}^{(i)}_{\mathsf{iter}}$ is set to 0, then output the shares obtained during the Π_{MultCI} instance.
  - **Negative Output**: If $\mathsf{flag}^{(i)}_{\mathsf{iter}}$ is set to 1 during the Π_{MultCI} instance, then proceed as follows.
    - **Identifying a Cheater Party Through ACS**: If $\mathsf{iter} = k \cdot (tn+1)$ for some $k \geq 1$, then do the following.
      1. Let $\mathcal{LD}^{(i)}_r$ be the set of locally-discarded parties for the instance $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], r)$, for $r = 1, \ldots, \mathsf{iter}$. For $j = 1, \ldots, n$, send $(\mathsf{vote}, \mathsf{sid}_{j,\mathsf{iter},k}, 1)$ to $\mathcal{F}_{\mathsf{ABA}}$ where $\mathsf{sid}_{j,\mathsf{iter},k} = \mathsf{sid}||j||\mathsf{iter}||k$, if for any $r \in \{1, \ldots, \mathsf{iter}\}$, party P_j is present in $\mathcal{LD}^{(i)}_r$ and $P_j \notin \mathcal{GD}$.
      2. For $j = 1, \ldots, n$, keep requesting for an output from $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{j,\mathsf{iter},k}$, until an output is received.
      3. Upon receiving $(\mathsf{decide}, \mathsf{sid}_{j,\mathsf{iter},k}, 1)$ from $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{j,\mathsf{iter},k}$ corresponding to some $P_j \in \mathcal{P}$, send $(\mathsf{vote}, \mathsf{sid}_{\ell,\mathsf{iter},k}, 0)$ to $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{\ell,\mathsf{iter},k}$ for each $P_\ell \in \mathcal{P}$, corresponding to which no vote message has been sent yet.
      4. Once an output $(\mathsf{decide}, \mathsf{sid}_{\ell,\mathsf{iter},k}, v_\ell)$ is received from $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{\ell,\mathsf{iter},k}$ for every $\ell \in \{1, \ldots, n\}$, select the minimum indexed party P_j from $\mathcal{P}$, such that $v_j = 1$. Then set $\mathcal{GD} = \mathcal{GD} \cup \{P_j\}$, set $\mathsf{iter} = \mathsf{iter} + 1$ and go to the step labelled **Multiplication with Cheater Identification**.
    - Else set $\mathsf{iter} = \mathsf{iter} + 1$ and go to the step **Multiplication with Cheater Identification**.

Figure 13: Multiplication protocol in the $(\mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{ABA}})$-hybrid for sid. The above code is executed by every party P_i

For a better understanding, a flowchart for the protocol Π_{Mult} is presented in Fig 14.

We next proceed to prove the properties of the protocol Π_{Mult}. While proving these properties, we will assume that $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition. This further implies that the sharing specification $\mathbb{S} = \{S_1, \ldots, S_h\} \stackrel{def}{=} \{\mathcal{P} \setminus Z | Z \in \mathcal{Z}\}$ satisfies the $\mathbb{Q}^{(3)}(\mathbb{S}, \mathcal{Z})$ condition.

We begin with the definition of a *successful iteration* in protocol Π_{Mult}.

Definition 3.28 (Successful Iteration). In protocol Π_{Mult}, an iteration iter is called *successful* if every honest P_i sets $\mathsf{flag}^{(i)}_{\mathsf{iter}} = 0$ during the corresponding instance $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], \mathsf{iter})$ of Π_{MultCI}.

We next claim that during each iteration of the protocol Π_{Mult}, the honest parties will know whether the iteration is successful or not.

Claim 3.29. For any iter, if all honest parties participate in iteration number iter of the protocol Π_{Mult} and if no honest party is ever included in the set $\mathcal{GD}$, then all honest parties will eventually agree on whether the iteration is successful or not.

Proof. We prove the claim through induction on iter. The statement is obviously true for $\mathsf{iter} = 1$, since during the instance $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], 1)$, all honest parties P_i will eventually set $\mathsf{flag}^{(i)}_1$ to a common

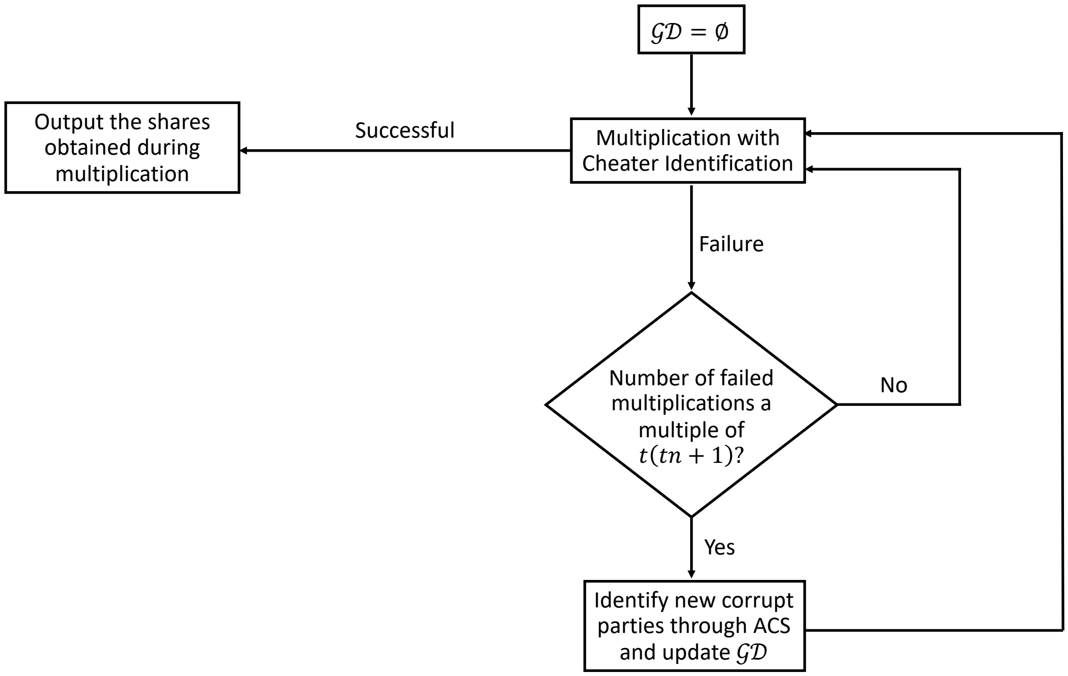

Figure 14: Flowchart for the protocol Π_{Mult}. Here t denotes the cardinality of the maximum-sized subset from $\mathcal{Z}$

value from $\{0, 1\}$ (follows from Lemma 3.27). Assume that the statement is true for $\mathsf{iter} = r$. Now consider $\mathsf{iter} = r + 1$ and let all honest parties participate in iteration number $r + 1$ by invoking the instance $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], r + 1)$. From the protocol steps, since the honest parties participate in iteration number $r + 1$, it implies that none of the previous r iterations were successful. From Lemma 3.27, all honest parties from the sets $\mathcal{W}_1^{(i)}, \ldots, \mathcal{W}_r^{(i)}$ will eventually be removed for every *honest* P_i. Moreover, no honest party will ever be included in the sets $\mathcal{LD}_1^{(i)}, \ldots, \mathcal{LD}_r^{(i)}$. Furthermore, as per the lemma condition, no honest party is ever included in the set $\mathcal{GD}$. It now follows from Claim 3.18 and Lemma 3.27 that during the instance $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], r + 1)$, all honest parties P_i will eventually set $\mathsf{flag}_{r+1}^{(i)}$ to a common value from $\{0, 1\}$ and learn whether the iteration is successful or not. $\square$

We next claim that if any iteration of Π_{Mult} is successful, then honest parties output $[ab]$ in that iteration.

Claim 3.30. If the iteration number iter in Π_{Mult} is successful, then honest parties output $[ab]$ during iteration number iter.

Proof. Let iteration number iter in Π_{Mult} be successful. This implies that every honest P_i sets $\mathsf{flag}_{\mathsf{iter}}^{(i)} = 0$ during the corresponding instance $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], \mathsf{iter})$ of Π_{MultCI} and hence this instance of Π_{MultCI} is successful. The proof now follows from Lemma 3.27. $\square$

We next prove that after every $tn + 1$ unsuccessful iterations of Π_{Mult}, a new corrupt party is globally discarded.

Claim 3.31. Let $t \stackrel{def}{=} \max\{|Z| : Z \in \mathcal{Z}\}$. In Π_{Mult}, for every $k \geq 1$, if none of the iterations $(k - 1)(tn + 1) + 1, \ldots, k(tn + 1)$ is successful, then eventually, a new corrupt party is included in the set $\mathcal{GD}$. Moreover, no honest party is ever included in the set $\mathcal{GD}$.

Proof. Let $Z^\star \in \mathcal{Z}$ be the set of corrupt parties during the execution of Π_{Mult}. We prove the claim through strong induction over k.

Base case: $k = 1$**.** We first note that from the protocol steps, the condition $\mathcal{GD} = \emptyset$ holds for each of the iterations $1, \ldots, tn+1$, during the corresponding instance of Π_{MultCI} in these iterations. Consequently, from Claim 3.29, for the iterations $1, \ldots, tn+1$, the honest parties agree on whether the iteration is successful or not. Let none of the iterations $1, \ldots, tn+1$ be successful. This implies that for $\mathsf{iter} = 1, \ldots, tn+1$, none of the instances $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], \mathsf{iter})$ of Π_{MultCI} is successful. This further implies that for every $\mathsf{iter} \in \{1, \ldots, tn+1\}$, there exists a well-defined *unordered* pair of parties (P_j, P_k), such that $f_{\mathsf{char}}(\mathsf{iter}) = (P_j, P_k)$, with at least one among P_j, P_k being *maliciously-corrupt* (follows from Claim 3.23). Let $\mathcal{C}$ denote the set of all pairs of "characteristic parties" for the first $tn+1$ instances of Π_{MultCI}. That is,

$$\mathcal{C} \stackrel{def}{=} \{(P_j, P_k) : f_{\mathsf{char}}(\mathsf{iter}) = (P_j, P_k) \text{ and } \mathsf{iter} \in \{1, \ldots, tn+1\}.$$

It then follows that $|\mathcal{C}| \leq tn$. This is because $|Z^\star| \leq t$, implying that there can be at most tn distinct (unordered) pairs of parties, where at least one of the parties in the pair is corrupt. Since the cardinality of $\mathcal{C}$ is smaller than the number of failed Π_{MultCI} instances, from the pigeonhole principle, we can conclude that there exist at least two iterations $r, r' \in \{1, \ldots, tn+1\}$ where $r < r'$, such that $f_{\mathsf{char}}(r) = f_{\mathsf{char}}(r') = (P_j, P_k)$.

Now, let us focus on the failed instances $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], r)$ and $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], r')$, corresponding to iteration number r and r' respectively in Π_{Mult}. Let $\mathcal{W}_r^{(i)}$ and $\mathcal{LD}_r^{(i)}$ be the dynamic sets maintained by every party P_i during the instance $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], r)$. Note that at the time of initializing $\mathcal{W}_r^{(i)}$, both P_j as well as P_k will be present in $\mathcal{W}_r^{(i)}$ (this follows from the protocol steps of Π_{MultCI}). Let $Z, Z' \in \mathcal{Z}$ be the *conflicting-sets* for the failed instance $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], r')$. From the definition of characteristic function f_{char}, it follows that $P_j, P_k \in \mathsf{Selected}_{(Z,r')} \cup \mathsf{Selected}_{(Z',r')}$. Hence, P_j (resp. P_k) is selected as a summand-sharing party in at least one of the instances $\Pi_{\mathsf{OptMult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], Z, r')$ or $\Pi_{\mathsf{OptMult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], Z', r')$. This further implies that there exists at least one honest party, say P_h, such that *both* P_j as well as P_k are removed by P_h from the set $\mathcal{W}_r^{(h)}$. This is because if both P_j as well as P_k are still present in the $\mathcal{W}_r^{(i)}$ set of *all* honest parties during the instances $\Pi_{\mathsf{OptMult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], Z, r')$ and $\Pi_{\mathsf{OptMult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], Z', r')$, then neither P_j not P_k will be selected as a summand-sharing party and hence $P_j, P_k \notin \mathsf{Selected}_{(Z,r')} \cup \mathsf{Selected}_{(Z',r')}$ (follows from Claim 3.16), which is a contradiction. Now, if both P_j and P_k are removed from $\mathcal{W}_r^{(h)}$, then from Claim 3.24, the corrupt party(ies) among P_j, P_k will be eventually included in the $\mathcal{LD}_r^{(i)}$ set of *every* honest P_i. For simplicity and without loss of generality, let P_j be the corrupt party among P_j, P_k.

In the protocol Π_{Mult}, once the parties find that iteration number $tn+1$ has failed, they run an instance of ACS to identify a cheating party across the first $tn+1$ failed instances, where the parties vote for candidate cheating parties based on the contents of their local $\mathcal{LD}$ sets. To complete the proof for the base case, we need to show that ACS will eventually output a common corrupt party for all the honest parties. The proof for this is similar to that of Claim 3.11. Namely, as argued above, the corrupt party P_j from the pair (P_j, P_k) above will be eventually included in the $\mathcal{LD}_r^{(i)}$ set of *every* honest P_i. We first show that there will be at least one instance of $\mathcal{F}_{\mathsf{ABA}}$, corresponding to which *all* honest parties eventually receive the output 1. For this, we consider two possible cases:

– *At least one honest party participates with input* 0 *in the* $\mathcal{F}_{\mathsf{ABA}}$ *instance corresponding to* P_j: Let P_i be an *honest* party, who sends $(\mathsf{vote}, \mathsf{sid}_{j,tn+1,1}, 0)$ to $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{j,tn+1,1}$. Then from the steps of Π_{Mult}, it follows that there exists some $P_\ell \in \mathcal{P}$, such that P_i has received $(\mathsf{decide}, \mathsf{sid}_{\ell,tn+1,1}, 1)$ as the output from $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{\ell,tn+1,1}$. Hence, *every* honest party will eventually receive the output $(\mathsf{decide}, \mathsf{sid}_{\ell,tn+1,1}, 1)$ as the output from $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{\ell,tn+1,1}$.

– *No honest party participates with input* 0 *in the* $\mathcal{F}_{\mathsf{ABA}}$ *instance corresponding to* P_j: In this case, *every* honest party will eventually send $(\mathsf{vote}, \mathsf{sid}_{j,tn+1,1}, 1)$ to $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{j,tn+1,1}$. This is because P_j will be eventually included in the $\mathcal{LD}_r^{(i)}$ set of *every* honest P_i. Consequently, every honest party eventually receives the output $(\mathsf{decide}, \mathsf{sid}_{j,tn+1,1}, 1)$ from $\mathcal{F}_{\mathsf{ABA}}$.

Now based on the above argument, we can further infer that all honest parties will eventually participate with some input in all the n instances of $\mathcal{F}_{\mathsf{ABA}}$ invoked after the first $tn+1$ failed iterations and hence, all the n instances of $\mathcal{F}_{\mathsf{ABA}}$ will eventually produce an output. Let P_m be the smallest indexed party such that $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{m,tn+1,1}$ has returned the output $(\mathsf{decide}, \mathsf{sid}_{m,tn+1,1}, 1)$. Hence, all honest parties eventually include P_m to $\mathcal{GD}$.

Finally, it is easy to see that $P_m \in Z^\star$. This is because if $P_m \notin Z^\star$, then P_m is *honest*. From Claim 3.19 it follows that P_m will *not* be included in the $\mathcal{LD}_{\mathsf{iter}}^{(i)}$ of any honest P_i for any $\mathsf{iter} \in \{1, \ldots, tn+1\}$. Consequently, *no* honest P_i will ever send $(\mathsf{vote}, \mathsf{sid}_{m,tn+1,1}, 1)$ to $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{m,tn+1,1}$. Hence, $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{m,tn+1,1}$ will never return the output $(\mathsf{decide}, \mathsf{sid}_{m,tn+1,1}, 1)$, which is a contradiction. This completes the proof for the base case.

Inductive Step. Let the statement be true for $k = 1, \ldots, k'$. Now consider the case when $k = k'+1$. Let $\mathcal{GD}_1, \ldots, \mathcal{GD}_{k'}$ be the set of discarded cheating parties after the iteration number $tn+1, \ldots, k'(tn+1)$ respectively.[15] From the inductive hypothesis, $\mathcal{GD}_1 \subset \mathcal{GD}_2 \subset \ldots \subset \mathcal{GD}_{k'}$ and no honest party is present in $\mathcal{GD}_{k'}$. Consequently, from the protocol steps and from Claim 3.29, for the iterations $k'(tn+1)+1, \ldots, (k'+1)(tn+1)$, the honest parties agree on whether the iteration is successful or not. Let none of the iterations $k'(tn+1)+1, \ldots, (k'+1)(tn+1)$ be successful. This implies that for $\mathsf{iter} = k'(tn+1)+1, \ldots, (k'+1)(tn+1)$, none of the instances $\Pi_{\mathsf{MultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], \mathsf{iter})$ of Π_{MultCI} is successful. In the protocol, once the parties find that the iteration number $(k'+1)(tn+1)$ is not successful, they proceed to select a common cheating party through ACS. Let $\mathcal{LD}_{\mathsf{iter}}^{(i)}, \mathcal{W}_{\mathsf{iter}}^{(i)}$ be the dynamic sets maintained by each party P_i across the iterations $1, \ldots, (k'+1)(tn+1)$.

We first note that none of the parties from $\mathcal{GD}_{k'}$ will be selected as a summand-sharing party in any of the underlying $\Pi_{\mathsf{OptMult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], Z, \mathsf{iter})$ instances, for any $\mathsf{iter} \in \{k'(tn+1)+1, \ldots, (k'+1)(tn+1)\}$ and any $Z \in \mathcal{Z}$ (this follows from Claim 3.16). We also note that there will be at least one party from $Z^\star$, which is not present in $\mathcal{GD}_{k'}$; i.e. $\mathcal{GD}_{k'} \subset Z^\star$. If not, then *only* honest parties will be selected as summand-sharing parties in all the underlying instances of Π_{OptMult} during the iteration number $k'(tn+1)+1$ and hence, the iteration number $k'(tn+1)+1$ in Π_{Mult} would be successful, which is a contradiction. Since the iteration number $k'(tn+1)+1, \ldots, (k'+1)(tn+1)$ constitutes $tn+1$ failed iterations, by applying the same pigeonhole-principle based argument as applied for the base case, we can infer that there exists a pair of iterations $r, r' \in \{k'(tn+1)+1, \ldots, (k'+1)(tn+1)\}$ where $r < r'$, such that $f_{\mathsf{char}}(r) = f_{\mathsf{char}}(r') = (P_j, P_k)$, with at least one among P_j and P_k being maliciously-corrupt. Moreover, the corrupt party(ies) among P_j, P_k will be from the set $Z^\star \setminus \mathcal{GD}_{k'}$, since the parties from $\mathcal{GD}_{k'}$ will *not* be selected as a summand-sharing party during the iteration number r and r'. Next, following the same argument as used for the base case, we can infer that the corrupt party(ies) among P_j and P_k will be eventually included in the $\mathcal{LD}_r^{(i)}$ set of every *honest* P_i. This will further imply all the n instances of $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{1,(k'+1)(tn+1),(k'+1)}, \ldots, \mathsf{sid}_{n,(k'+1)(tn+1),(k'+1)}$ will eventually return an output for all the honest parties, such that at least one of the $\mathcal{F}_{\mathsf{ABA}}$ instances with $\mathsf{sid}_{\ell,(k'+1)(tn+1),(k'+1)}$ corresponding to the party P_ℓ will return an output $(\mathsf{decide}, \mathsf{sid}_{\ell,(k'+1)(tn+1),(k'+1)}, 1)$. Let P_m be the smallest indexed party corresponding to which the $\mathcal{F}_{\mathsf{ABA}}$ instance with $\mathsf{sid}_{m,(k'+1)(tn+1),(k'+1)}$ returns the output $(\mathsf{decide}, \mathsf{sid}_{m,(k'+1)(tn+1),(k'+1)}, 1)$. Hence the honest parties will update $\mathcal{GD}$ to $\mathcal{GD}_{k'} \cup \{P_m\}$. To complete the proof, we need to show that $P_m \notin$

[15] Recall that in the protocol, ACS is executed after every block of $tn+1$ failed iterations and $\mathcal{GD}$ gets updated through ACS. In the context of the given scenario, the parties would have run ACS after iteration numbers $tn+1, 2(tn+1), \ldots, (k'-1)(tn+1)$ and $k'(tn+1)$ to update the set $\mathcal{GD}$. The set $\mathcal{GD}_{k'}$ denotes the updated $\mathcal{GD}$ set after the ACS instance number k'.

$\mathcal{GD}_{k'}$ and $P_m \in Z^\star$. The former follows from the fact that if $P_m \in \mathcal{GD}_{k'}$, then it implies that then no honest party ever sends $(\mathsf{vote}, \mathsf{sid}_{m,(k'+1)(tn+1),(k'+1)}, 1)$ to $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{m,(k'+1)(tn+1),(k'+1)}$ and consequently, $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{m,(k'+1)(tn+1),(k'+1)}$ will never return the output $(\mathsf{decide}, \mathsf{sid}_{m,(k'+1)(tn+1),(k'+1)}, 1)$. On the other hand, $P_m \in Z^\star$ follows using a similar argument as used for the base case. □

An immediate corollary of Claim 3.31 is that there can be at most $t(tn+1)$ consecutive failed iterations in the protocol Π_{Mult}.

Corollary 3.32. *In protocol Π_{Mult}, there can be at most $t(tn+1)$ consecutive failed iterations, where $t \stackrel{def}{=} \max\{|Z| : Z \in \mathcal{Z}\}$.*

We next claim that it will take at most $t(tn+1)+1$ iterations in the protocol Π_{Mult} to guarantee that there is at least one successful iteration.

Claim 3.33. In protocol Π_{Mult}, it will take at most $t(tn+1)+1$ iterations to ensure that one of these iterations is successful.

Proof. Follows easily from Claim 3.29 and Corollary 3.32. □

We next claim that the adversary does not learn anything additional about a and b in the protocol.

Claim 3.34. In protocol Π_{Mult}, Adv does not learn anything additional about a and b.

Proof. Follows directly from the fact that in every iteration of Π_{Mult}, Adv does not learn anything additional about a and b, which in turn follows from Claim 3.25. □

Lemma 3.35 now follows from Claim 3.33, Claim 3.30 and Claim 3.34, where the communication complexity follows from the communication complexity of Π_{MultCI} and the fact that there are $t(tn+1)+1 = \mathcal{O}(n^3)$ instances of Π_{MultCI} executed inside the protocol Π_{Mult}.

Lemma 3.35. *Let $\mathcal{Z}$ satisfy the $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition and let $\mathbb{S} = \{\mathcal{P} \setminus Z | Z \in \mathcal{Z}\}$. Then Π_{Mult} takes at most $t(tn+1)$ iterations and all honest parties eventually output a secret-sharing of $[ab]$, where $t = \max\{|Z| : Z \in \mathcal{Z}\}$. In the protocol,* Adv *does not learn anything additional about a and b. The protocol makes $\mathcal{O}(|\mathcal{Z}| \cdot n^5)$ calls to $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$ and additionally incurs a communication of $\mathcal{O}(|\mathcal{Z}|^2 \cdot n^5 \log |\mathbb{F}| + |\mathcal{Z}| \cdot n^7 \log |\mathbb{F}|)$ bits.*

Protocol Π_{Mult} for M Pairs of Inputs. The modifications to the protocol Π_{Mult} for handling M pairs of secret-shared inputs $\{[a^{(\ell)}], [b^{(\ell)}]\}_{\ell=1,\ldots,M}$ are simple. Now, the instances of Π_{MultCI} are executed with M pairs of inputs in each iteration. This requires $\mathcal{O}(M \cdot |\mathcal{Z}| \cdot n^5)$ calls to $\mathcal{F}_{\mathsf{VSS}}$, $\mathcal{O}(|\mathcal{Z}| \cdot n^5)$ calls to $\mathcal{F}_{\mathsf{ABA}}$ and additional communication of $\mathcal{O}((M \cdot |\mathcal{Z}|^2 \cdot n^5 + |\mathcal{Z}| \cdot n^7) \log |\mathbb{F}|)$ bits.

3.3 The Pre-Processing Phase Protocol

The perfectly-secure pre-processing phase protocol $\Pi_{\mathsf{PerTriples}}$ is standard. We first explain the protocol for generating a random secret-sharing of one multiplication-triple. The modifications to generate M secret-shared multiplication-triples are straightforward.

The protocol consists of two stages. In the first stage, the parties jointly generate a secret-sharing of a random pair of values and during the second stage, the parties securely generate a secret-sharing of the product of these values. For the first stage, each party secret-shares a pair of random values. The parties then run an instance of ACS to agree on a common subset of parties whose selected pair of values are eventually secret-shared, such that an *honest* party is guaranteed to be included in the subset. The sum of the secret-sharing of the pairs of values shared by the parties in the subset is considered the output of the first stage. For the second stage, the parties simply run an instance of Π_{Mult} over the output pair of secret-shared values obtained at the end of the first stage.

Protocol $\Pi_{\mathsf{PerTriples}}$ for securely realizing $\mathcal{F}_{\mathsf{Triples}}$ with $M = 1$ in the $(\mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{ABA}})$-hybrid model is formally presented in Fig 15.

Protocol $\Pi_{\mathsf{PerTriples}}(\mathcal{P}, \mathcal{Z}, \mathbb{S})$

– **Stage I: Generating a Secret-Sharing of Random Pair of Values**.
  - **Sharing Random Pairs of Values**:
    1. Randomly select $a^{(i)}, b^{(i)} \in \mathbb{F}$ and shares $(a_1^{(i)}, \ldots, a_h^{(i)})$ and $(b_1^{(i)}, \ldots, b_h^{(i)})$, such that $a_1^{(i)} + \ldots + a_h^{(i)} = a^{(i)}$ and $b_1^{(i)} + \ldots + b_h^{(i)} = b^{(i)}$. Call $\mathcal{F}_{\mathsf{VSS}}$ with $(\mathsf{dealer}, \mathsf{sid}_{i,1}, (a_1^{(i)}, \ldots, a_h^{(i)}))$ and $\mathcal{F}_{\mathsf{VSS}}$ with $(\mathsf{dealer}, \mathsf{sid}_{i,2}, (b_1^{(i)}, \ldots, b_h^{(i)}))$ for $\mathsf{sid}_{i,1}$ and $\mathsf{sid}_{i,2}$, where $\mathsf{sid}_{i,1} = \mathsf{sid}||i||1$ and $\mathsf{sid}_{i,2} = \mathsf{sid}||i||2$.
    2. For $j = 1, \ldots, n$, keep requesting for an output from $\mathcal{F}_{\mathsf{VSS}}$ with $\mathsf{sid}_{j,1}$ and $\mathsf{sid}_{j,2}$, till an output is received.
  - **Selecting a Common Subset of Parties Through ACS**
    1. Upon receiving $(\mathsf{share}, \mathsf{sid}_{j,1}, P_j, \{[a^{(j)}]_q\}_{P_i \in S_q})$ and $(\mathsf{share}, \mathsf{sid}_{j,2}, P_j, \{[b^{(j)}]_q\}_{P_i \in S_q})$ from $\mathcal{F}_{\mathsf{VSS}}$ with $\mathsf{sid}_{j,1}$ and $\mathsf{sid}_{j,2}$ respectively, send $(\mathsf{vote}, \mathsf{sid}_j, 1)$ to $\mathcal{F}_{\mathsf{ABA}}$, where $\mathsf{sid}_j \stackrel{def}{=} \mathsf{sid}||j$.
    2. For $j = 1, \ldots, n$, request for output from $\mathcal{F}_{\mathsf{ABA}}$ with sid_j, till an output is received.
    3. Upon receiving $(\mathsf{decide}, \mathsf{sid}_j, 1)$ from $\mathcal{F}_{\mathsf{ABA}}$ with sid_j corresponding to every $P_j \in \mathcal{GP}_i$ for any subset of parties $\mathcal{GP}_i$ such that $\mathcal{P} \setminus \mathcal{GP}_i \in \mathcal{Z}$, send $(\mathsf{vote}, \mathsf{sid}_j, 0)$ to $\mathcal{F}_{\mathsf{ABA}}$ with sid_j corresponding to every P_j, for which no message has been sent yet.
    4. Once $(\mathsf{decide}, \mathsf{sid}_j, v_j)$ is received from $\mathcal{F}_{\mathsf{ABA}}$ for $j = 1, \ldots, n$, set $\mathcal{CS} = \{P_j : v_j = 1\}$.
    5. Let $a \stackrel{def}{=} \sum_{P_j \in \mathcal{CS}} a^{(j)}, b \stackrel{def}{=} \sum_{P_j \in \mathcal{CS}} b^{(j)}$. Locally compute the shares $\{[a]_q\}_{P_i \in S_q}$ and $\{[b]_q\}_{P_i \in S_q}$.

– **Stage II: Generating the Product**.
  - Participate in the instance $\Pi_{\mathsf{Mult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b])$ with sid and compute $\{[c]_q\}_{P_i \in S_q}$. Output $\{[a]_q, [b]_q, [c]_q\}_{P_i \in S_q}$.

Figure 15: A perfectly-secure protocol to securely realize $\mathcal{F}_{\mathsf{Triples}}$ with $M = 1$ in $(\mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{ABA}})$-hybrid model for session id sid. The above code is executed by every party P_i

Protocol $\Pi_{\mathsf{PerTriples}}$ for Generating M Multiplication-Triples. The modifications in $\Pi_{\mathsf{PerTriples}}$ to generate M multiplication-triples are straight forward. During the first stage, each party secret-shares M pairs of values, by calling $\mathcal{F}_{\mathsf{VSS}}$ $2M$ number of times. While running ACS, a party votes "positively" for party P_j, only after receiving an output from *all* the $2M$ instances of $\mathcal{F}_{\mathsf{VSS}}$ corresponding to P_j. During the second stage, the instance of Π_{Mult} will now take M pairs of secret-shared inputs.

We now prove the security of the protocol $\Pi_{\mathsf{PerTriples}}$ in the $(\mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{ABA}})$-hybrid model. While proving these properties, we will assume that $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition. This further implies that the sharing specification $\mathbb{S} = \{S_1, \ldots, S_h\} \stackrel{def}{=} \{\mathcal{P} \setminus Z | Z \in \mathcal{Z}\}$ satisfies the $\mathbb{Q}^{(3)}(\mathbb{S}, \mathcal{Z})$ condition.

Theorem 3.36. *If $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition, then $\Pi_{\mathsf{PerTriples}}$ is a perfectly-secure protocol for securely realizing $\mathcal{F}_{\mathsf{Triples}}$ with UC-security in the $(\mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{ABA}})$-hybrid model. The protocol makes $\mathcal{O}(M \cdot |\mathcal{Z}| \cdot n^5)$ calls to $\mathcal{F}_{\mathsf{VSS}}$, $\mathcal{O}(|\mathcal{Z}| \cdot n^5)$ calls to $\mathcal{F}_{\mathsf{ABA}}$ and additionally incurs a communication of $\mathcal{O}(M \cdot |\mathcal{Z}|^2 \cdot n^5 \log |\mathbb{F}| + |\mathcal{Z}| \cdot n^7 \log |\mathbb{F}|)$ bits.*

Proof. The communication complexity and the number of calls to $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$ follow from the protocol steps and the communication complexity of the protocol Π_{Mult}. So we next prove the security. For ease of explanation, we consider the case where only one multiplication-triple is generated in $\Pi_{\mathsf{PerTriples}}$; i.e. $M = 1$. The proof can be easily modified for any general M.

Let Adv be an arbitrary adversary, attacking the protocol $\Pi_{\mathsf{PerTriples}}$ by corrupting a set of parties $Z^\star \in \mathcal{Z}$, and let Env be an arbitrary environment. We show the existence of a simulator $\mathcal{S}_{\mathsf{PerTriples}}$ (Fig 16), such that for any $Z^\star \in \mathcal{Z}$, the outputs of the honest parties and the view of the adversary in the protocol $\Pi_{\mathsf{PerTriples}}$ is indistinguishable from the outputs of the honest parties and the view of the adversary in an execution in the ideal world involving $\mathcal{S}_{\mathsf{PerTriples}}$ and $\mathcal{F}_{\mathsf{Triples}}$.

The high level idea of the simulator is as follows. Throughout the simulation, the simulator itself performs the role of the ideal functionality $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$ whenever required. During the first stage, whenever Adv sends a pair of vector of shares to $\mathcal{F}_{\mathsf{VSS}}$ on the behalf of a corrupt party, the simulator records these vectors. On the other hand, for the honest parties, the simulator picks pairs of random values and random shares for those values, and distributes the appropriate shares to the corrupt parties, as per $\mathcal{F}_{\mathsf{VSS}}$. During ACS, to select the common subset of parties, the simulator itself performs the role of $\mathcal{F}_{\mathsf{ABA}}$ and simulates the honest parties as per the steps of the protocol and $\mathcal{F}_{\mathsf{ABA}}$. This allows the simulator learn the common subset of parties $\mathcal{CS}$. Notice that the secret-sharing of the pairs of values shared by *all* the parties in $\mathcal{CS}$ will be available with the simulator. While the secret-sharing of pairs of the honest parties in $\mathcal{CS}$ are selected by the simulator itself, for every *corrupt* party P_j which is added to $\mathcal{CS}$, at least one honest party P_i should participate with input 1 in the corresponding call to $\mathcal{F}_{\mathsf{ABA}}$. This implies that the honest party P_i must have received some shares from $\mathcal{F}_{\mathsf{VSS}}$ corresponding to the vector of shares which P_j sent to $\mathcal{F}_{\mathsf{VSS}}$. Since in the simulation, the role of $\mathcal{F}_{\mathsf{VSS}}$ is played by the simulator itself, it implies that the vector of shares used by P_j will be known to the simulator.

Once the simulator learns $\mathcal{CS}$ and the secret-sharing of the pairs of values shared by the parties in $\mathcal{CS}$, during the second stage, the simulator simulates the rest of the interaction between the honest parties and Adv as per the protocol steps, by itself playing the role of the honest parties. Moreover, in the underlying instances of Π_{OptMult}, Π_{MultCI} and Π_{Mult}, the simulator itself performs the role of $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$ whenever required. Notice that simulator will be knowing the values which should be shared by the respective parties through $\mathcal{F}_{\mathsf{VSS}}$ during the underlying instances of Π_{OptMult} and Π_{MultCI}. This is because these values are completely determined by the secret-sharing of the pairs of values shared by the parties in $\mathcal{CS}$, which will be known to the simulator. Consequently, in the simulated execution, the simulator will be knowing which instances of Π_{MultCI} are successful and which iterations of Π_{Mult} are successful. Once the simulated execution is over, the simulator learns the shares of the corrupt parties corresponding to the output multiplication-triple in the simulated execution. The simulator then communicates these shares on the behalf of the corrupt parties during its interaction with $\mathcal{F}_{\mathsf{Triples}}$. This ensures that the shares of the corrupt parties remain the same in both the worlds.

In the steps of the simulator, to distinguish between the values used by the various parties during the real execution and simulated execution, the variables in the simulated execution will be used with a $\tilde{}$ symbol.

Simulator $\mathcal{S}_{\mathsf{PerTriples}}$

$\mathcal{S}_{\mathsf{PerTriples}}$ constructs virtual real-world honest parties and invokes the real-world adversary Adv. The simulator simulates the view of Adv, namely its communication with Env, the messages sent by the honest parties, and the interaction with $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$. In order to simulate Env, the simulator $\mathcal{S}_{\mathsf{PerTriples}}$ forwards every message it receives from Env to Adv and vice-versa. The simulator then simulates the various stages of the protocol as follows.

- **Stage I: Generating a Secret-Sharing of a Random Pair of Values**.
  - **Sharing Random Pairs of Values**:
    - The simulator simulates the operations of the honest parties during this phase by picking random pairs of values and random vector of shares for those values on their behalf. Namely, when Adv

requests for output from $\mathcal{F}_{\mathsf{VSS}}$ with $\mathsf{sid}_{j,1}$ and $\mathsf{sid}_{j,2}$ for any $P_j \notin Z^\star$, the simulator picks random values $\widetilde{a}^{(j)}, \widetilde{b}^{(j)} \in \mathbb{F}$ and random shares $(\widetilde{a}^{(j)}_1, \ldots, \widetilde{a}^{(j)}_h)$ and $(\widetilde{b}^{(j)}_1, \ldots, \widetilde{b}^{(j)}_h)$, such that $\widetilde{a}^{(j)}_1 + \ldots + \widetilde{a}^{(j)}_h = \widetilde{a}^{(j)}$ and $\widetilde{b}^{(j)}_1 + \ldots + \widetilde{b}^{(j)}_h = \widetilde{b}^{(j)}$. The simulator then sets $[\widetilde{a}^{(j)}]_q = \widetilde{a}^{(j)}_q$ and $[\widetilde{b}^{(j)}]_q = \widetilde{b}^{(j)}_q$ for $q = 1, \ldots, h$, and responds to Adv with the output $(\mathsf{share}, \mathsf{sid}_{j,1}, P_j, \{[\widetilde{a}^{(j)}]_q\}_{S_q \cap Z^\star \neq \emptyset})$ and $(\mathsf{share}, \mathsf{sid}_{j,2}, P_j, \{[\widetilde{b}^{(j)}]_q\}_{S_q \cap Z^\star \neq \emptyset})$ on the behalf of $\mathcal{F}_{\mathsf{VSS}}$ with $\mathsf{sid}_{j,1}$ and $\mathsf{sid}_{j,2}$ respectively.

- Whenever Adv sends $(\mathsf{dealer}, \mathsf{sid}_{i,1}, (\widetilde{a}^{(i)}_1, \ldots, \widetilde{a}^{(i)}_h))$ and $(\mathsf{dealer}, \mathsf{sid}_{i,2}, (\widetilde{b}^{(i)}_1, \ldots, \widetilde{b}^{(i)}_h))$ to $\mathcal{F}_{\mathsf{VSS}}$ with $\mathsf{sid}_{i,1}$ and $\mathsf{sid}_{i,2}$ respectively on the behalf of any $P_i \in Z^\star$, the simulator sets $[\widetilde{a}^{(i)}]_q = \widetilde{a}^{(i)}_q$ and $[\widetilde{b}^{(i)}]_q = \widetilde{b}^{(i)}_q$ for $q = 1, \ldots, h$, where $\widetilde{a}^{(i)} \stackrel{def}{=} \widetilde{a}^{(i)}_1 + \ldots + \widetilde{a}^{(i)}_h$ and $\widetilde{b}^{(i)} \stackrel{def}{=} \widetilde{b}^{(i)}_1 + \ldots + \widetilde{b}^{(i)}_h$.

- **Selecting a Common Subset of Parties (ACS)**: The simulator simulates the interface to $\mathcal{F}_{\mathsf{ABA}}$ for Adv by itself performing the role of $\mathcal{F}_{\mathsf{ABA}}$ and playing the role of the honest parties, as per the steps of the protocol. When the first honest party completes this phase during the simulated execution, $\mathcal{S}_{\mathsf{PerTriples}}$ learns the set $\mathcal{CS}$. It then sets $\widetilde{a} \stackrel{def}{=} \sum_{P_j \in \mathcal{CS}} \widetilde{a}^{(j)}, \widetilde{b} \stackrel{def}{=} \sum_{P_j \in \mathcal{CS}} \widetilde{b}^{(j)}$, computes $[\widetilde{a}] = \sum_{P_j \in \mathcal{CS}} [\widetilde{a}^{(j)}]$ and $[\widetilde{b}] = \sum_{P_j \in \mathcal{CS}} [\widetilde{b}^{(j)}]$.

– **Stage II: Generating the Product**. The simulator plays the role of the honest parties as per the protocol and interacts with Adv for the instance $\Pi_{\mathsf{Mult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [\widetilde{a}], [\widetilde{b}])$, where during the instance, the simulator uses the shares $\{[\widetilde{a}]_q, [\widetilde{b}]_q\}_{P_i \in S_q}$ on the behalf of every $P_i \notin Z^\star$. Moreover, during this instance of Π_{Mult}, the simulator simulates the interface to $\mathcal{F}_{\mathsf{ABA}}$ for Adv during the underlying instances of Π_{OptMult} and during cheater identification, by itself performing the role of $\mathcal{F}_{\mathsf{ABA}}$, as per the steps of the protocol. Furthermore, during the underlying instances of Π_{OptMult} and Π_{MultCI}, whenever required, the simulator itself plays the role $\mathcal{F}_{\mathsf{VSS}}$.

– **Interaction with** $\mathcal{F}_{\mathsf{Triples}}$: Let $\{[\widetilde{c}]_q\}_{S_q \cap Z^\star \neq \emptyset}$ be the output shares of the parties in $Z^\star$, during the instance $\Pi_{\mathsf{Mult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [\widetilde{a}], [\widetilde{b}])$. The simulator sends $(\mathsf{shares}, \mathsf{sid}, \{[\widetilde{a}]_q, [\widetilde{b}]_q, [\widetilde{c}]_q\}_{S_q \cap Z^\star \neq \emptyset})$ to $\mathcal{F}_{\mathsf{Triples}}$, on the behalf of the parties in $Z^\star$.

Figure 16: Simulator for the protocol $\Pi_{\mathsf{PerTriples}}$ with $M = 1$ where Adv corrupts the parties in set $Z^\star \in \mathcal{Z}$

We now prove a series of claims, which will help us to finally prove the theorem. We first claim that in any execution of $\Pi_{\mathsf{PerTriples}}$, a set $\mathcal{CS}$ is eventually generated.

Claim 3.37. In any execution of $\Pi_{\mathsf{PerTriples}}$, a common set $\mathcal{CS}$ is eventually generated where $\mathcal{P} \setminus \mathcal{CS} \in \mathcal{Z}$, such that for every $P_j \in \mathcal{CS}$, there exists a pair of values held by P_j, which are eventually secret-shared.

Proof. We first show that there always exists some set $Z \in \mathcal{Z}$ such that in the $\mathcal{F}_{\mathsf{ABA}}$ instances corresponding to every party in $\mathcal{P} \setminus Z$, all honest parties eventually obtain an output 1. For this, we consider the following two cases.

– *If some honest party P_i has participated with* vote *input* 0 *in any instance of* $\mathcal{F}_{\mathsf{ABA}}$ *during step 3 of the ACS phase*: this implies that there exists a subset $\mathcal{GP}_i$ for P_i where $\mathcal{P} \setminus \mathcal{GP}_i \in \mathcal{Z}$, such that P_i receives the output $(\mathsf{decide}, \mathsf{sid}_j, 1)$ from $\mathcal{F}_{\mathsf{ABA}}$ with sid_j, corresponding to every $P_j \in \mathcal{GP}_i$. Consequently, every honest party will eventually receive the same outputs from these $\mathcal{F}_{\mathsf{ABA}}$ instances. Since $\mathcal{P} \setminus \mathcal{GP}_i \in \mathcal{Z}$, we get that there exists some set $Z \in \mathcal{Z}$ such that the $\mathcal{F}_{\mathsf{ABA}}$ instances corresponding to every party in $\mathcal{P} \setminus Z$ responded with output 1, which is what we wanted to show.

– *No honest party has participated with* vote *input* 0 *in any instance of* $\mathcal{F}_{\mathsf{ABA}}$: In the protocol, each party $P_j \notin Z^\star$ sends its vector of shares to $\mathcal{F}_{\mathsf{VSS}}$ with $\mathsf{sid}_{j,1}$ and $\mathsf{sid}_{j,2}$ and every *honest* party eventually receives its respective shares from these vectors as the output from the corresponding instances of $\mathcal{F}_{\mathsf{VSS}}$. Hence, corresponding to every $P_j \notin Z^\star$, *all* honest parties eventually participate with input $(\mathsf{vote}, \mathsf{sid}_j, 1)$ during the instance of $\mathcal{F}_{\mathsf{ABA}}$ with sid_j, and this $\mathcal{F}_{\mathsf{ABA}}$ instance will eventually respond with output $(\mathsf{decide}, \mathsf{sid}_j, 1)$. Since $Z^\star \in \mathcal{Z}$, it then follows that even in this case, there exists some set $Z \in \mathcal{Z}$ such that the $\mathcal{F}_{\mathsf{ABA}}$ instances corresponding to every party in $\mathcal{P} \setminus Z$ responded with output 1.

We next show that all honest parties eventually receive an output from *all* the instances of $\mathcal{F}_{\mathsf{ABA}}$. Since we have shown there exists some set $Z \in \mathcal{Z}$ such that the $\mathcal{F}_{\mathsf{ABA}}$ instances corresponding to every party in $\mathcal{P} \setminus Z$ eventually returns the output 1, it thus follows that all honest parties eventually participate with some vote inputs in the remaining $\mathcal{F}_{\mathsf{ABA}}$ instances and hence will eventually obtain some output from these $\mathcal{F}_{\mathsf{ABA}}$ instances as well. Since the set $\mathcal{CS}$ corresponds to the $\mathcal{F}_{\mathsf{ABA}}$ instances in which the honest parties obtain 1 as the output, it thus follows that eventually, the honest parties obtain some $\mathcal{CS}$ where $\mathcal{P} \setminus \mathcal{CS} \in \mathcal{Z}$. Moreover, the set $\mathcal{CS}$ will be common, as it is based on the outcome of $\mathcal{F}_{\mathsf{ABA}}$ instances.

Now consider an arbitrary $P_j \in \mathcal{CS}$. This implies that the parties obtain 1 as the output from the j^{th} instance of $\mathcal{F}_{\mathsf{ABA}}$. This further implies that at least one *honest* party P_i participated in this $\mathcal{F}_{\mathsf{ABA}}$ instance with vote input 1. This is possible only if P_i received its respective shares from the instances of $\mathcal{F}_{\mathsf{VSS}}$ with $\mathsf{sid}_{j,1}$ and $\mathsf{sid}_{j,2}$, further implying that P_j has provided some vector of shares $(a_1^{(j)}, \ldots, a_h^{(j)})$ and $(b_1^{(i)}, \ldots, b_h^{(i)})$ as inputs to these $\mathcal{F}_{\mathsf{VSS}}$ instances. It now follows easily that eventually, all honest parties will have their respective shares corresponding to the vectors of shares provided by P_j, implying that the honest parties will eventually hold $[a^{(j)}]$ and $[b^{(j)}]$, where $a^{(j)} \stackrel{def}{=} a_1^{(j)} + \ldots + a_h^{(j)}$ and $b^{(j)} \stackrel{def}{=} b_1^{(j)} + \ldots + b_h^{(j)}$. $\square$

We next show that the view generated by $\mathcal{S}_{\mathsf{PerTriples}}$ for Adv is identically distributed to Adv's view during the real execution of $\Pi_{\mathsf{PerTriples}}$.

Claim 3.38. The view of Adv in the simulated execution with $\mathcal{S}_{\mathsf{PerTriples}}$ is identically distributed as the view of Adv in the real execution of $\Pi_{\mathsf{PerTriples}}$.

Proof. We first note that in the real-world (during the real execution of $\Pi_{\mathsf{PerTriples}}$), the view of Adv consists of the following:

(1) The vector of shares $(a_1^{(j)}, \ldots, a_h^{(j)})$ and $(b_1^{(i)}, \ldots, b_h^{(i)})$ (if any) for $\mathcal{F}_{\mathsf{VSS}}$ with $\mathsf{sid}_{j,1}$ and $\mathsf{sid}_{j,2}$ respectively, corresponding to $P_j \in Z^\star$.
(2) Shares $\{[a^{(j)}]_q, [b^{(j)}]_q\}_{S_q \cap Z^\star \neq \emptyset}$, corresponding to $P_j \notin Z^\star$.
(3) Inputs of the various parties during the $\mathcal{F}_{\mathsf{ABA}}$ instances as part of ACS and the outputs from the $\mathcal{F}_{\mathsf{ABA}}$ instances.
(4) The view generated for Adv during the instance of Π_{Mult}.

The vectors of shares in (1) are the inputs of Adv and hence they are identically distributed in both the real as well as simulated execution of $\Pi_{\mathsf{PerTriples}}$, so let us fix these vectors. In the real execution, every $P_j \notin Z^\star$ picks its pair of values randomly and the vectors of shares for $\mathcal{F}_{\mathsf{VSS}}$, corresponding to these values, uniformly at random. On the other hand, in the simulated execution, the simulator picks the pair of values and their shares randomly on the behalf of P_j. Now since the sharing specification $\mathbb{S} = (S_1, \ldots, S_h) \stackrel{def}{=} \{\mathcal{P} \setminus Z | Z \in \mathcal{Z}\}$ is $\mathcal{Z}$*-private*, it follows that the distribution of the shares in (2) is identical in both the real, as well as the simulated execution. Specifically, conditioned on the shares in (2), the underlying pairs of values shared by the parties $P_j \notin Z^\star$ are uniformly distributed. Since the partial view of Adv containing (1) and (2) are identically distributed, let us fix them.

Now conditioned on (1) and (2), it is easy to see that the partial view of Adv consisting of (3) is identically distributed in both the executions. This is because the outputs of the $\mathcal{F}_{\mathsf{ABA}}$ instances are determined *deterministically* based on the inputs provided by the various parties in these $\mathcal{F}_{\mathsf{ABA}}$ instances. Furthermore, the inputs of the parties in these $\mathcal{F}_{\mathsf{ABA}}$ instances depend upon the order in which various parties receive outputs from various $\mathcal{F}_{\mathsf{VSS}}$ instances, which is completely determined by Adv, since message scheduling is under the control of Adv. Since in the simulated execution, the simulator provides the interface to various instances of $\mathcal{F}_{\mathsf{ABA}}$ to Adv in exactly the same way as $\mathcal{F}_{\mathsf{ABA}}$ would have been accessed by Adv in the real execution, it follows that the partial view of Adv containing $(1), (2)$ and (3) is identically distributed in both the executions and so let us fix this. This also fixes the set $\mathcal{CS}$, which according to Claim 3.37, is guaranteed to be generated.

Let $[a]$ and $[b]$ be the secret-sharing held by the honest parties after stage I, conditioned on the view of Adv in $(1), (2)$ and (3). Note that in the simulated execution, the simulator will be knowing the complete sharing $[a]$ and $[b]$. This is because $[a]$ and $[b]$ are computed *deterministically* based on the secret-sharing of the pairs of the values shared by the parties in $\mathcal{CS}$, all of which will be completely known to the simulator in the simulated execution. To complete the proof of the claim, we need to show that the partial view of Adv consisting of (4) is identically distributed in both the executions (conditioned on $(1), (2)$ and (3)). However, this follows from the privacy of $\Pi_{\mathsf{OptMult}}, \Pi_{\mathsf{MultCI}}$ and Π_{Mult} (Claims 3.14, 3.25 and 3.34) and the fact that in the simulated execution, simulator plays the role of the honest parties during the instance of Π_{Mult} exactly as per the steps of Π_{Mult}, where the simulator will be completely knowing the shares of both $[a]$ and $[b]$ corresponding to both the honest as well as corrupt parties. Consequently, it will be knowing the shares with which different parties have to participate in the underlying instances of Π_{OptMult} and Π_{MultCI}. Moreover, in the simulated execution, the simulator honestly plays the role of $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$ in these Π_{OptMult} and Π_{MultCI} instances. This guarantees that the view of Adv during the real execution of the Π_{Mult} instance is exactly the same as the view of Adv during the simulated execution of Π_{Mult}. □

Finally, we show that conditioned on the view of Adv, the outputs of the honest parties are identically distributed in both the worlds.

Claim 3.39. Conditioned on the view of Adv, the output of the honest parties are identically distributed in the real execution of $\Pi_{\mathsf{PerTriples}}$ involving Adv, as well as in the ideal execution involving $\mathcal{S}_{\mathsf{PerTriples}}$ and $\mathcal{F}_{\mathsf{Triples}}$.

Proof. Let View be an arbitrary view of Adv, and let $\{([a^{(j)}], [b^{(j)}])\}_{P_j \in \mathcal{CS}}$ be the secret-sharing of the pairs of values as per View, shared by the parties in $\mathcal{CS}$. Note that $\mathcal{CS}$ is bound to have at least one honest party. This is because $\mathcal{P} \setminus \mathcal{CS} \in \mathcal{Z}$ and if $\mathcal{CS} \subseteq Z^{\star}$, then it implies that $\mathcal{Z}$ *does not* satisfy the $\mathbb{Q}^{(2)}(\mathcal{P}, \mathcal{Z})$ condition, which is a contradiction. From the proof of Claim 3.38, it follows that corresponding to every *honest* $P_j \in \mathcal{CS}$, the pairs $(a^{(j)}, b^{(j)})$ are uniformly distributed conditioned on the shares of these pairs, as determined by View. Let us fix these pairs.

We show that in the real-world, the honest parties eventually output $([a], [b], [c])$, where conditioned on View, the triple (a, b, c) is a uniformly random multiplication-triple over $\mathbb{F}$. From the protocol steps, the parties set $[a] \stackrel{def}{=} \sum_{P_j \in \mathcal{CS}} [a^{(j)}], [b] \stackrel{def}{=} \sum_{P_j \in \mathcal{CS}} [b^{(j)}]$. Since corresponding to every $P_j \in \mathcal{CS}$, the honest parties eventually hold $[a^{(j)}]$ and $[b^{(j)}]$ (follows from Claim 3.37), it follows that the honest parties eventually hold $[a]$ and $[b]$. Moreover, since $[c]$ is computed as the output of the instance $\Pi_{\mathsf{Mult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b])$, it follows from Lemma 3.35 that the honest parties will eventually hold $[c]$, where $c = ab$. We next show that conditioned on View, the multiplication-triple (a, b, c) is uniformly distributed over $\mathbb{F}$. However, this follows from the fact that there exists a one-to-one correspondence between the random pairs shared by the honest parties in $\mathcal{CS}$ and (a, b). Namely, from the viewpoint of Adv, for every candidate pair $(a^{(j)}, b^{(j)})$ shared by the *honest* parties $P_j \in \mathcal{CS}$, there exists a unique (a, b) which is consistent with View. Since the pairs shared by the *honest* parties P_j are uniformly distributed and independent of View, it follows that (a, b) is also uniformly distributed. Since $c = ab$ holds, it follows that (a, b, c) is uniformly distributed.

To complete the proof, we now show that conditioned on the shares $\{([a]_q, [b]_q, [c]_q)\}_{S_q \cap Z^{\star} \neq \emptyset}$ (which are determined by View), the honest parties output a secret-sharing of some random multiplication-triple in the ideal-world which is consistent with the shares $\{([a]_q, [b]_q, [c]_q)\}_{S_q \cap Z^{\star} \neq \emptyset}$. However, this simply follows from the fact that in the ideal-world, the simulator $\mathcal{S}_{\mathsf{PerTriples}}$ sends the shares $\{([a]_q, [b]_q, [c]_q)\}_{S_q \cap Z^{\star} \neq \emptyset}$ to

$\mathcal{F}_{\mathsf{Triples}}$ on the behalf of the parties in $Z^{\star}$. As an output, $\mathcal{F}_{\mathsf{Triples}}$ generates a random secret-sharing of some random multiplication-triple consistent with the shares provided by $\mathcal{S}_{\mathsf{PerTriples}}$. $\square$

The proof of Theorem 3.36 now follows from Claim 3.38 and Claim 3.39. $\square$

4 Statistically-Secure Pre-Processing Phase Protocol with $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ Condition

In this section, we present a statistically-secure protocol for securely realizing $\mathcal{F}_{\mathsf{Triples}}$, provided $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. We first design a statistically-secure VSS protocol, for which we present an AICP with $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition.

4.1 Asynchronous Information Checking Protocol (AICP)

An ICP [32, 16] is used for authenticating data in the presence of a *computationally-unbounded* adversary. An AICP [10, 29] extends ICP for the *asynchronous* setting. In an AICP, there are *four* entities, a *signer* $\mathsf{S} \in \mathcal{P}$, an *intermediary* $\mathsf{I} \in \mathcal{P}$, a *receiver* $\mathsf{R} \in \mathcal{P}$ and all the parties in $\mathcal{P}$ acting as *verifiers* (note that S, I and R also act as verifiers). An AICP has two sub-protocols, one for the *authentication phase* and one for the *revelation phase*.

In the authentication phase, S has some private input $s \in \mathbb{F}$, which it distributes to I along with some *authentication information*. Each verifier is provided with some *verification information*, followed by the parties verifying whether S has distributed consistent information. The data held by I at the end of this phase is called S*'s IC-Signature on s for intermediary* I *and receiver* R, denoted by $\mathsf{ICSig}(\mathsf{S}, \mathsf{I}, \mathsf{R}, s)$. Later, during the revelation phase, I reveals $\mathsf{ICSig}(\mathsf{S}, \mathsf{I}, \mathsf{R}, s)$ to R, who "verifies" it with respect to the verification information provided by the verifiers and decides whether to accept or reject s. We require the same security guarantees from AICP as expected from digital signatures, namely *correctness*, (if S, I and R are *all* honest, then R should accept s), *unforgeability* (a *corrupt* I should fail to reveal an *honest* S's signature on $s' \neq s$) and *non-repudiation* (if an *honest* I holds some $\mathsf{ICSig}(\mathsf{S}, \mathsf{I}, \mathsf{R}, s)$, then later an *honest* R should accept s, even if S is *corrupt*). Additionally, we will need the *privacy* property guaranteeing that if S, I and R are *all* honest, then Adv does not learn s.

Our AICP is a generalization of the AICP of [29], which was designed against *threshold* adversaries. During the authentication phase, S embeds s in a random t-degree *signing-polynomial* $F(x)$, where t is the cardinality of maximum-sized subset in $\mathcal{Z}$, and gives $F(x)$ to I. In addition, each verifier P_i is given a random *verification-point* (α_i, v_i) on $F(x)$. Later, during the revelation phase, I is supposed to reveal $F(x)$ to R, while each verifier P_i is supposed to reveal their verification-points to R. R then accepts $F(x)$ if it is found to be consistent with "sufficiently many" verification-points. The above idea achieves all the properties of AICP, except the *non-repudiation* property, since a potentially *corrupt* S may distribute "inconsistent" data to I and the verifiers. To deal with this, during the authentication phase, the parties interact in a "zero-knowledge" fashion to verify the consistency of the distributed information. For this, S additionally distributes a random t-degree *masking-polynomial* $M(x)$ to I, while each verifier P_i is given a point on $M(x)$ at α_i. The parties then publicly check the consistency of the $F(x), M(x)$ polynomials and the distributed points with respect to a random linear combination of these polynomials and points. The linear combiner is randomly selected by I only when it is confirmed that S has distributed the verification-points to sufficiently many *supporting verifiers* in a set $\mathcal{SV}$. This ensures that S has no knowledge beforehand about the random combiner while distributing the points to $\mathcal{SV}$ and hence, any inconsistency among the data distributed by a *corrupt* S will be detected with a high probability.

Protocol AICP

Protocol $\Pi_{\mathsf{Auth}}(\mathcal{P}, \mathcal{Z}, \mathsf{S}, \mathsf{I}, \mathsf{R}, s)$

– **Distributing the Polynomials and the Verification Points**: Only S executes the following steps.
  - Randomly select t-degree *signing-polynomial* $F(x)$ and *masking-polynomial* $M(x)$, such that $F(0) = s$, where $t = \max\{|Z| : Z \in \mathcal{Z}\}$.
  - For $j = 1, \ldots, n$, randomly select $\alpha_j \in \mathbb{F} \setminus \{0\}$ and compute $v_j = F(\alpha_j), m_j = M(\alpha_j)$.
  - Send $(\mathsf{authPoly}, \mathsf{sid}, F(x), M(x))$ to I.
  - For $j = 1, \ldots, n$, send $(\mathsf{authPoint}, \mathsf{sid}, (\alpha_j, v_j, m_j))$ to party P_j.

– **Confirming Receipt of Verification Points**: Each party P_i (including S, I and R) upon receiving $(\mathsf{authPoint}, \mathsf{sid}, (\alpha_i, v_i, m_i))$ from S, sends $(\mathsf{Received}, \mathsf{sid}, i)$ to I.

– **Announcing Masked Polynomial and Set of Supporting Verifiers**:
  - I, upon receiving $(\mathsf{Received}, \mathsf{sid}, j)$ from a set of parties $\mathcal{SV}$ where $\mathcal{P} \setminus \mathcal{SV} \in \mathcal{Z}$, randomly picks $d \in \mathbb{F} \setminus \{0\}$ and sends $(\mathsf{sender}, \mathsf{Acast}, \mathsf{sid}_{\mathsf{I}}, (d, B(x), \mathcal{SV}))$ to $\mathcal{F}_{\mathsf{Acast}}$, where $\mathsf{sid}_{\mathsf{I}} = \mathsf{sid}||\mathsf{I}$ and $B(x) \stackrel{def}{=} dF(x) + M(x)$.
  - Every party $P_i \in \mathcal{P}$ keeps requesting for output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{\mathsf{I}}$ until an output is received.

– **Announcing Validity of Masked Polynomial:**
  - S, upon receiving an output $(\mathsf{I}, \mathsf{Acast}, \mathsf{sid}_{\mathsf{I}}, (d, B(x), \mathcal{SV}))$ from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{\mathsf{I}}$, checks if $B(x)$ is a t-degree polynomial, $\mathcal{P} \setminus \mathcal{SV} \in \mathcal{Z}$ and $dv_j + m_j = B(\alpha_j)$ holds for all $P_j \in \mathcal{SV}$.
    – If all the above conditions hold, then S sends $(\mathsf{sender}, \mathsf{Acast}, \mathsf{sid}_{\mathsf{S}}, \mathsf{OK})$ to $\mathcal{F}_{\mathsf{Acast}}$, where $\mathsf{sid}_{\mathsf{S}} = \mathsf{sid}||\mathsf{S}$;
    – Else, S sends $(\mathsf{sender}, \mathsf{Acast}, \mathsf{sid}_{\mathsf{S}}, \mathsf{NOK}, s)$ to $\mathcal{F}_{\mathsf{Acast}}$.
  - Every party $P_i \in \mathcal{P}$ keeps requesting for output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{\mathsf{S}}$ until an output is received.

– **Deciding Whether Authentication is Successful**: Every party P_i (including S, I and R) upon receiving $(\mathsf{I}, \mathsf{Acast}, \mathsf{sid}_{\mathsf{I}}, (d, B(x), \mathcal{SV}))$ from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{\mathsf{I}}$, sets the variable $\mathsf{authCompleted}_{\mathsf{S,I,R}}^{(\mathsf{sid},i)}$ to 1 if either of the following holds.
  - $(\mathsf{sender}, \mathsf{Acast}, \mathsf{sid}_{\mathsf{S}}, \mathsf{NOK}, s)$ is received from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{\mathsf{S}}$. In this case, P_i also sets $\mathsf{ICSig}(\mathsf{S}, \mathsf{I}, \mathsf{R}, s) = s$.
  - $(\mathsf{sender}, \mathsf{Acast}, \mathsf{sid}_{\mathsf{S}}, \mathsf{OK})$ is received from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{\mathsf{S}}$. Here, P_i sets $\mathsf{ICSig}(\mathsf{S}, \mathsf{I}, \mathsf{R}, s) = F(x)$, if $P_i = \mathsf{I}$.[a]

Protocol $\Pi_{\mathsf{Reveal}}(\mathcal{P}, \mathcal{Z}, \mathsf{S}, \mathsf{I}, \mathsf{R}, s)$

– **Revealing Signing Polynomial and Verification Points**: Each party P_i (including S, I and R) does the following, if $\mathsf{authCompleted}_{\mathsf{S,I,R}}^{(\mathsf{sid},i)}$ is set to 1 and $\mathsf{ICSig}(\mathsf{S}, \mathsf{I}, \mathsf{R}, s)$ has *not* been *publicly* set during Π_{Auth}.
  - If $P_i = \mathsf{I}$ then send $(\mathsf{revealPoly}, \mathsf{sid}, F(x))$ to R, where $\mathsf{ICSig}(\mathsf{S}, \mathsf{I}, \mathsf{R}, s)$ has been set to $F(x)$ during Π_{Auth}.
  - If $P_i \in \mathcal{SV}$, then send $(\mathsf{revealPoint}, \mathsf{sid}, (\alpha_i, v_i, m_i))$ to R.

– **Accepting or Rejecting the IC-Sig**: The following steps are executed only by R, if $\mathsf{authCompleted}_{\mathsf{S,I,R}}^{(\mathsf{sid},i)}$ is set to 1 by R during the protocol $\Pi_{\mathsf{Auth}}(\mathcal{P}, \mathcal{Z}, \mathsf{S}, \mathsf{I}, \mathsf{R}, s)$, where $\mathsf{R} = P_i$.
  – If R has set $\mathsf{ICSig}(\mathsf{S}, \mathsf{I}, \mathsf{R}, s) = s$ during Π_{Auth}, then output s.
  – Else, wait till $(\mathsf{revealPoly}, \mathsf{sid}, F(x))$ is received from I, where $F(x)$ is a t-degree polynomial. Upon receiving, proceed as follows.
    1. Upon receiving $(\mathsf{revealPoint}, \mathsf{sid}, (\alpha_j, v_j, m_j))$ from any $P_j \in \mathcal{SV}$, *accept* (α_j, v_j, m_j) if either of the following holds:
       – $v_j = F(\alpha_j)$; or
       – $B(\alpha_j) \neq dv_j + m_j$, where $B(x)$ is received from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{\mathsf{I}}$, during Π_{Auth}.
    2. Wait till a subset of parties $\mathcal{SV}' \subseteq \mathcal{SV}$ is found, such that $\mathcal{SV} \setminus \mathcal{SV}' \in \mathcal{Z}$ and for every $P_j \in \mathcal{SV}'$, the corresponding revealed point (α_j, v_j, m_j) is accepted. Then output $s = F(0)$.

[a] If S broadcasts s along with NOK, then ICSig will be set *publicly* to s, while if S broadcasts OK then *only* I sets ICSig to $F(x)$.

Figure 17: The asynchronous information-checking protocol against general adversaries for session id sid in the $\mathcal{F}_{\mathsf{Acast}}$-hybrid

We next formally prove the properties of our AICP. While proving these properties, we assume that $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. We first show that when S, I and R are honest, all honest parties set the local bit indicating that the authentication has completed to 1. Furthermore, R will accept the signature revealed by I.

Claim 4.1 (Correctness). If S, I and R are *honest*, then each honest P_i eventually sets $\mathsf{authCompleted}_{\mathsf{S},\mathsf{I},\mathsf{R}}^{(\mathsf{sid},i)}$ to 1 during Π_{Auth}. Moreover, R eventually outputs s during Π_{Reveal}.

Proof. Let S, I and R be honest and let $\mathcal{H}$ be the set of *honest* parties among $\mathcal{P}$. Moreover, let $Z^\star \in \mathcal{Z}$ be the set of *corrupt* parties. We first show that each honest party P_i eventually sets $\mathsf{authCompleted}_{\mathsf{S},\mathsf{I},\mathsf{R}}^{(\mathsf{sid},i)}$ to 1 during Π_{Auth}. During Π_{Auth}, S chooses the signing-polynomial $F(x)$ such that $s = F(0)$ holds. S will then send the signing-polynomial $F(x)$ and masking-polynomial $M(x)$ to I, and the corresponding verification-point (α_i, v_i, m_i) to each verifier P_i, such that $v_i = F(\alpha_i)$ and $m_i = M(\alpha_i)$ holds. Consequently, each verifier in $\mathcal{H}$ will eventually receive its verification-point and indicates this to I. Since $\mathcal{P} \setminus \mathcal{H} = Z^\star \in \mathcal{Z}$, it follows that I will eventually find a set $\mathcal{SV}$, such that $\mathcal{P} \setminus \mathcal{SV} \in \mathcal{Z}$, and where each verifier in $\mathcal{SV}$ has indicated to I that it has received its verification-point. Consequently, I will compute $B(x) = dF(x)+M(x)$, and broadcast $(d, B(x), \mathcal{SV})$, which is eventually delivered to every honest party, including S. Moreover, S will find that $B(\alpha_j) = dv_j + m_j$ holds for all the verifiers $P_j \in \mathcal{SV}$. Consequently, S will broadcast an OK message, which is eventually received by every honest party P_i, who then sets $\mathsf{authCompleted}_{\mathsf{S},\mathsf{I},\mathsf{R}}^{(\mathsf{sid},i)}$ to 1. Moreover, I will set $\mathsf{ICSig}(\mathsf{S}, \mathsf{I}, \mathsf{R}, s)$ to $F(x)$.

During Π_{Reveal}, I will send $F(x)$ to R, and each verifier $P_i \in \mathcal{H} \cap \mathcal{SV}$ will send its verification points (α_i, v_i, m_i) to R. These points and the polynomial $F(x)$ are eventually received by R. Moreover, the condition $v_i = F(\alpha_i)$ will hold true for these points, and consequently these points will be *accepted*. Since $\mathcal{SV} \setminus (\mathcal{H} \cap \mathcal{SV}) \subseteq Z^\star \in \mathcal{Z}$, it follows that R will eventually find a subset $\mathcal{SV}' \subseteq \mathcal{SV}$ where $\mathcal{SV} \setminus \mathcal{SV}' \in \mathcal{Z}$, such that the points corresponding to all the parties in $\mathcal{SV}'$ are accepted. This implies that R will eventually output $s = F(0)$. ☐

We next show that when S, I and R are *honest*, the adversary does not learn anything about s during either Π_{Auth} or Π_{Reveal}.

Claim 4.2 (Privacy). If S, I and R are *honest*, then the view of adversary Adv throughout Π_{Auth} and Π_{Reveal} is independent of s.

Proof. Let $t = \max\{|Z| : Z \in \mathcal{Z}\}$ and let $Z^\star \in \mathcal{Z}$ be the set of corrupt parties. For simplicity and without loss of generality, let $|Z^\star| = t$. During Π_{Auth}, the adversary Adv learns t verification-points $\{(\alpha_i, v_i, m_i)\}_{P_i \in Z^\star}$. However, since $F(x)$ is a random t-degree polynomial with $F(0) = s$, the points $\{(\alpha_i, v_i)\}_{P_i \in Z^\star}$ are distributed independently of s. That is, for every candidate $s \in \mathbb{F}$ from the point of view of Adv, there is a corresponding unique t-degree polynomial $F(x)$, such that $F(\alpha_i) = v_i$ holds corresponding to every $P_i \in Z^\star$.

During Π_{Auth}, the adversary Adv also learns d and the blinded-polynomial $B(x) = dF(x) + M(x)$, along with the points $\{(\alpha_i, v_i)\}_{P_i \in Z^\star}$. However, this does not add any new information about s to the view of the adversary. This is because $M(x)$ is a random t-degree polynomial. Hence for every candidate $M(x)$ polynomial from the point of view of Adv where $M(\alpha_i) = m_i$ holds for every $P_i \in Z^\star$, there is a corresponding unique t-degree polynomial $F(x)$, such that $F(\alpha_i) = v_i$ holds corresponding to every $P_i \in Z^\star$, and where $dF(x) + M(x) = B(x)$. We also note that in Π_{Auth}, the signer S does not broadcast s, which follows from the Claim 4.1. Finally, Adv does not learn anything new about s during Π_{Reveal}, since the verification-points and the signing-polynomial are sent only to R. ☐

We next prove the unforgeability property.

Claim 4.3 (Unforgeability). If S, R are *honest*, I is corrupt and if R outputs s' during Π_{Reveal}, then $s' = s$ holds, except with probability at most $\frac{nt}{|\mathbb{F}|-1}$.

Proof. Let $\mathcal{H}$ be the set of honest parties in $\mathcal{P}$ and let $Z^\star$ be the set of corrupt parties. Since R outputs s' during Π_{Reveal}, it implies that during Π_{Auth}, the variable $\mathsf{authCompleted}_{\mathsf{S},\mathsf{I},\mathsf{R}}^{(\mathsf{sid},i)}$ is set to 1 by R, if $\mathsf{R} = P_i$. This further implies that S has broadcasted either an OK or an NOK message during Π_{Auth}, which further implies that I has broadcasted some blinded-polynomial $B(x)$ during Π_{Auth}. Now there are now two possible cases.

– S *has broadcasted* NOK *along with* s *during* Π_{Auth}: In this case, every honest party including R would set $\mathsf{ICSig}(\mathsf{S}, \mathsf{I}, \mathsf{R}, s)$ to s during Π_{Auth}. Moreover, during Π_{Reveal}, the receiver R outputs s. Hence, in this case, $s' = s$ holds with probability 1.
– S *has broadcasted* OK *during* Π_{Auth}: This implies that during Π_{Auth}, I had broadcasted a t-degree blinded-polynomial $B(x)$, along with the set $\mathcal{SV}$. Furthermore, S has verified that $\mathcal{P} \setminus \mathcal{SV} \in \mathcal{Z}$ and $B(\alpha_i) = dv_i + m_i$ holds for every verifier $P_i \in \mathcal{SV}$. Now during Π_{Reveal}, if I sends $F(x)$ as $\mathsf{ICSig}(\mathsf{S}, \mathsf{I}, \mathsf{R}, s)$ to R, then again $s' = s$ holds with probability 1. So consider the case when I sends $F'(x)$ as $\mathsf{ICSig}(\mathsf{S}, \mathsf{I}, \mathsf{R}, s)$ to R, where $F'(x)$ is a t-degree polynomial such that $F'(x) \neq F(x)$ and where $F'(0) = s'$. In this case, we show that except with probability at most $\frac{nt}{|\mathbb{F}|-1}$, the verification-point of no *honest* verifier from $\mathcal{SV}$ will get accepted by R during Π_{Reveal}, with respect to $F'(x)$. Now assuming that this statement is true, the proof follows from the fact that in order for $F'(x)$ to be accepted by R, it should accept the verification-point of at least one *honest* verifier from $\mathcal{SV}$ with respect to $F'(x)$. This is because R should find a subset of verifiers $\mathcal{SV}' \subseteq \mathcal{SV}$ whose corresponding verification-points are accepted, where $\mathcal{SV} \setminus \mathcal{SV}' \in \mathcal{Z}$. So clearly, the set of *corrupt* verifiers in $\mathcal{SV}$ *cannot* form a candidate $\mathcal{SV}'$. This is because since $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition, it satisfies the $\mathbb{Q}^{(2)}(\mathcal{SV}, \mathcal{Z})$ condition as $\mathcal{P} \setminus \mathcal{SV} \in \mathcal{Z}$. This further implies that $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(1)}(\mathcal{SV}', \mathcal{Z})$ condition as $\mathcal{SV} \setminus \mathcal{SV}' \in \mathcal{Z}$. Hence, any candidate for $\mathcal{SV}'$ must contain at least one honest party from $\mathcal{SV}$.
  Consider an arbitrary verifier $P_i \in \mathcal{H} \cap \mathcal{SV}$ from which R receives the verification-point (α_i, v_i, m_i) during Π_{Reveal}. This point can be *accepted* only if either of the following holds.
  - $v_i = F'(\alpha_i)$: This is possible with probability at most $\frac{t}{|\mathbb{F}|-1}$. This is because $F'(x)$ and $F(x)$, being distinct t-degree polynomials can have at most t points in common, and since the evaluation-point α_i corresponding to P_i, being randomly selected from $\mathbb{F} - \{0\}$, will not be known to I.
  - $dv_i + m_i \neq B(\alpha_i)$: This is impossible, as otherwise S would have broadcasted s and NOK during Π_{Auth}, which is a contradiction.

  Now as there could be up to $n-1$ *honest* verifiers in $\mathcal{SV}$, it follows from the union bound that except with probability at most $\frac{nt}{|\mathbb{F}|-1}$, the polynomial $F'(x)$ will not be accepted.

□

We next prove the non-repudiation property.

Claim 4.4 (Non-Repudiation). If S is *corrupt* and I, R are *honest* and if I has set $\mathsf{ICSig}(\mathsf{S}, \mathsf{I}, \mathsf{R}, s)$ during Π_{Auth}, then R eventually outputs s during Π_{Reveal}, except with probability at most $\frac{n}{|\mathbb{F}|-1}$.

Proof. Let $\mathcal{H}$ be the set of honest parties in $\mathcal{P}$ and $Z^\star \in \mathcal{Z}$ be the set of corrupt parties. Since I has set $\mathsf{ICSig}(\mathsf{S}, \mathsf{I}, \mathsf{R}, s)$ during Π_{Auth}, it implies that that it has set the variable $\mathsf{authCompleted}_{\mathsf{S},\mathsf{I},\mathsf{R}}^{(\mathsf{sid},i)}$ to 1 during Π_{Auth}, if $\mathsf{I} = P_i$. This further implies that I has broadcasted a blinded-polynomial $B(x)$, the linear combiner d and the set $\mathcal{SV}$, where $B(x) = dF(x) + M(x)$ and where $F(x)$ and $M(x)$ are the signing and masking polynomials received by I from S. Moreover, S has broadcasted either an OK message or an NOK message.

Consequently, all *honest* parties P_j, including R, eventually set $\mathsf{authCompleted}_{\mathsf{S},\mathsf{I},\mathsf{R}}^{(\mathsf{sid},j)}$ to 1. Now there are two possible cases.

- S *has broadcasted* NOK, *along with* s *during* Π_{Auth}: In this case, all *honest* parties, including I and R, set $\mathsf{ICSig}(\mathsf{S}, \mathsf{I}, \mathsf{R}, s)$ to s during Π_{Auth}. Moreover, from the steps of Π_{Reveal}, R outputs s during Π_{Reveal}. Thus, the claim holds in this case with probability 1.
- S *has broadcasted* OK *during* Π_{Auth}: In this case, I sets $\mathsf{ICSig}(\mathsf{S}, \mathsf{I}, \mathsf{R}, s)$ to $F(x)$, where $F(0) = s$. During Π_{Reveal}, I sends $F(x)$ to R. Moreover, every verifier $P_i \in \mathcal{H} \cap \mathcal{SV}$ eventually sends its verification-point (α_i, v_i, m_i) to R. We next show that except with probability at most $\frac{n}{|\mathbb{F}|-1}$, all these verification-points are accepted by R. Now assuming that this statement is true, the proof follows from the fact that $\mathcal{H} \cap \mathcal{SV} = \mathcal{SV} \setminus Z^{\star}$. Consequently, R eventually accepts the verification-points from a subset of parties $\mathcal{SV}' \subseteq \mathcal{SV}$ where $\mathcal{SV} \setminus \mathcal{SV}' \in \mathcal{Z}$ and outputs s.

  Consider an arbitrary verifier $P_i \in \mathcal{H} \cap \mathcal{SV}$ whose verification-point (α_i, v_i, m_i) is received by R during Π_{Reveal}. Now there are two possible cases, depending upon the relationship that holds between $F(\alpha_i)$ and v_i during Π_{Auth}.
  - $v_i = F(\alpha_i)$ *holds*: In this case, according to the protocol steps of Π_{Reveal}, the point (α_i, v_i, m_i) is *accepted* by R.
  - $v_i \neq F(\alpha_i)$ *holds*: In this case, we claim that except with probability at most $\frac{1}{|\mathbb{F}|-1}$, the condition $dv_i + m_i \neq B(\alpha_i)$ will hold, implying that the point (α_i, v_i, m_i) is *accepted* by R. This is because the *only* way $dv_i + m_i = B(\alpha_i)$ holds is when S distributes (α_i, v_i, m_i) to P_i where $v_i \neq F(\alpha_i)$ and $m_i \neq M(\alpha_i)$ holds, and I selects $d = (M(\alpha_i) - m_i) \cdot (v_i - F(\alpha_i))^{-1}$. However, S will *not* be knowing the random d from $\mathbb{F} \setminus \{0\}$ which I picks while distributing $F(x), M(x)$ to I, and (α_i, v_i, m_i) to P_i.

  Now, as there can be up to $n-1$ *honest* verifiers in $\mathcal{SV}$, from the union bound, it follows that except with probability at most $\frac{n}{|\mathbb{F}|-1}$, the verification-point of *all* honest verifiers in $\mathcal{SV}$ are accepted by R. □

We next derive the communication complexity of Π_{Auth} and Π_{Reveal}.

Claim 4.5. Protocol Π_{Auth} incurs a communication of $\mathcal{O}(n \cdot \log |\mathbb{F}|)$ bits and makes $\mathcal{O}(1)$ calls to $\mathcal{F}_{\mathsf{Acast}}$ with $\mathcal{O}(n \cdot \log |\mathbb{F}|)$-bit messages. Protocol Π_{Reveal} requires a communication of $\mathcal{O}(n \cdot \log |\mathbb{F}|)$ bits.

Proof. During Π_{Auth}, signer S sends t-degree polynomials $F(x)$ and $M(x)$ to I, and verification-points to each verifier. This requires a communication of $\mathcal{O}(n \cdot \log |\mathbb{F}|)$ bits. Intermediary I needs to broadcast $B(x), d$ and the set $\mathcal{SV}$, which requires one call to $\mathcal{F}_{\mathsf{Acast}}$ with a message of size $\mathcal{O}(n \cdot \log |\mathbb{F}|)$ bits. Moreover, S may need to broadcast s, which requires one call to $\mathcal{F}_{\mathsf{Acast}}$ with a message of size $\mathcal{O}(\log |\mathbb{F}|)$ bits. During Π_{Reveal}, I may send $F(x)$ to R, and each verifier may send its verification-point to R. This will require a communication of $\mathcal{O}(n \cdot \log |\mathbb{F}|)$ bits. □

Lemma 4.6 now follows from Claims 4.1-4.5.

Lemma 4.6. *Let* $\mathcal{Z}$ *satisfy the* $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ *condition. Then the pair of protocols* $(\Pi_{\mathsf{Auth}}, \Pi_{\mathsf{Reveal}})$ *satisfy the following properties, except with probability at most* $\epsilon_{\mathsf{AICP}} \stackrel{def}{=} \frac{nt}{|\mathbb{F}|-1}$, *where* $t = \max\{|Z| : Z \in \mathcal{Z}\}$.

- **Correctness***: If* S, I *and* R *are honest, then each honest* P_i *eventually sets* $\mathsf{authCompleted}_{\mathsf{S},\mathsf{I},\mathsf{R}}^{(\mathsf{sid},i)}$ *to* 1 *during* Π_{Auth}. *Moreover,* R *eventually outputs* s *during* Π_{Reveal}.
- **Privacy***: If* S, I *and* R *are honest, then the view of adversary remains independent of* s.
- **Unforgeability***: If* S, R *are honest,* I *is corrupt and if* R *outputs* $s' \in \mathbb{F}$ *during* Π_{Reveal}, *then* $s' = s$ *holds.*
- **Non-repudiation***: If* S *is corrupt and* I, R *are honest and if* I *has set* $\mathsf{ICSig}(\mathsf{S}, \mathsf{I}, \mathsf{R}, s)$ *during* Π_{Auth}, *then* R *eventually outputs* s *during* Π_{Reveal}.

Protocol Π_{Auth} *requires a communication of* $\mathcal{O}(n \cdot \log|\mathbb{F}|)$ *bits and makes* $\mathcal{O}(1)$ *calls to* $\mathcal{F}_{\mathsf{Acast}}$ *with* $\mathcal{O}(n \cdot \log|\mathbb{F}|)$*-bit messages. Protocol* Π_{Reveal} *requires a communication of* $\mathcal{O}(n \cdot \log|\mathbb{F}|)$ *bits.*

Before proceeding further, we introduce certain notations for our AICP, which will be used when we use our AICP in our statistically-secure VSS protocol.

Notation 4.7 (Notation for Using AICP). We use the following terms while invoking $(\Pi_{\mathsf{Auth}}, \Pi_{\mathsf{Reveal}})$:

– "P_i *gives* $\mathsf{ICSig}(\mathsf{sid}, P_i, P_j, P_k, s)$ *to* P_j" to mean that P_i acts as S and invokes an instance of the protocol Π_{Auth} with session id sid, where P_j and P_k plays the role of I and R respectively.

– "P_j *receives* $\mathsf{ICSig}(\mathsf{sid}, P_i, P_j, P_k, s)$ *from* P_i" to mean that P_j, as I, has set $\mathsf{authCompleted}^{(\mathsf{sid},j)}_{P_i,P_j,P_k}$ to 1 during protocol Π_{Auth} with session id sid, where P_i and P_k plays the role of S and R respectively.

– "P_j *reveals* $\mathsf{ICSig}(\mathsf{sid}, P_i, P_j, P_k, s)$ *to* P_k" to mean P_j, as I, invokes an instance of Π_{Reveal} with session id sid, with P_i and P_k playing the role of S and R respectively.

– "P_k *accepts* $\mathsf{ICSig}(\mathsf{sid}, P_i, P_j, P_k, s)$" to mean that P_k, as R, outputs s during the instance of Π_{Reveal} with session id sid, invoked by P_j as I, with P_i playing the role of S.

4.2 Statistically-Secure VSS Protocol with $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ Condition

The high level idea behind our statistically-secure protocol Π_{SVSS} (Figure 18) is similar to that of the *perfectly-secure* VSS protocol Π_{PVSS} (see Fig 6). In Π_{PVSS}, dealer P_{D}, on having the shares $(s_1, \ldots, s_h)$, sends s_q to the parties in $S_q \in \mathbb{S}$. This is followed by the parties in S_q performing pair-wise consistency tests of their supposedly common shares and publicly announcing the results. Based on these results, the parties identify a *core* set $\mathcal{C}_q \subseteq S_q$, where $S_q \setminus \mathcal{C}_q \in \mathcal{Z}$, such that all the (honest) parties in $\mathcal{C}_q$ have received the same share s_q from P_D. Once such a $\mathcal{C}_q$ is identified, then the *honest* parties in $\mathcal{C}_q$, forming a "majority", can "help" the (honest) parties in $S_q \setminus \mathcal{C}_q$ get this common s_q. However, since $\mathcal{Z}$ *now* satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition, $\mathcal{C}_q$ may have *only one* honest party. Consequently, the "majority-based filtering" used by the parties in $S_q \setminus \mathcal{C}_q$ to get s_q will fail.

To deal with the above problem, the parties in S_q issue IC-Signatures during the pair-wise consistency tests of their supposedly common shares. The parties then check whether the common share s_q held by the (honest) parties in $\mathcal{C}_q$ is "(P_i, P_j, P_k)-authenticated" for *every* $P_i, P_j \in \mathcal{C}_q$ and *every* $P_k \in S_q$; i.e. P_j holds $\mathsf{ICSig}(P_i, P_j, P_k, s_q)$. Now, to help the parties $P_k \in S_q \setminus \mathcal{C}_q$ obtain the common share s_q, *every* $P_j \in \mathcal{C}_q$ reveals IC-signed s_q to P_k, signed by *every* $P_i \in \mathcal{C}_q$. Since $\mathcal{C}_q$ is bound to contain at least one *honest* party, a *corrupt* P_j will *fail* to forge an *honest* P_i's IC-signature on an incorrect s_q. On the other hand, an *honest* P_j will be *able* to eventually reveal the IC-signature of *all* the parties in $\mathcal{C}_q$ on the share s_q, which is accepted by P_k. Protocol Π_{SVSS} is formally presented in Fig 18.

Protocol $\Pi_{\mathsf{SVSS}}(\mathbb{S})$

– **Distribution of Shares**: P_{D}, on having input $(s_1, \ldots, s_h)$, sends $(\mathsf{dist}, \mathsf{sid}, q, s_q)$ to all $P_i \in S_q$, for $q = 1, \ldots, h$.

– **Pairwise Consistency Tests on IC-Signed Values**: For each $S_q \in \mathbb{S}$, each $P_i \in S_q$ does the following.
  - Upon receiving $(\mathsf{dist}, \mathsf{sid}, q, s_{qi})$ from D, give $\mathsf{ICSig}(\mathsf{sid}^{(P_{\mathsf{D}},q)}_{i,j,k}, P_i, P_j, P_k, s_{qi})$ to every $P_j \in S_q$, corresponding to every $P_k \in S_q$, where $\mathsf{sid}^{(P_{\mathsf{D}},q)}_{i,j,k} = \mathsf{sid}||P_{\mathsf{D}}||q||i||j||k$.
  - Upon receiving $\mathsf{ICSig}(\mathsf{sid}^{(P_{\mathsf{D}},q)}_{j,i,k}, P_j, P_i, P_k, s_{qj})$ from $P_j \in S_q$ corresponding to every party $P_k \in S_q$, if $s_{qi} = s_{qj}$ holds, then send $(\mathsf{sender}, \mathsf{Acast}, \mathsf{sid}^{(P_{\mathsf{D}},q)}_{i,j}, \mathsf{OK}_q(i,j))$ to $\mathcal{F}_{\mathsf{Acast}}$, where $\mathsf{sid}^{(P_{\mathsf{D}},q)}_{i,j} = \mathsf{sid}||P_{\mathsf{D}}||q||i||j$.

– **Constructing Consistency Graph**: For each $S_q \in \mathbb{S}$, each $P_i \in \mathcal{P}$ executes the following steps.
  - Initialize a set $\mathcal{C}_q$ to $\emptyset$. Construct an undirected consistency graph $G^{(i)}_q$ with S_q as the vertex set.
  - For every $P_j, P_k \in S_q$, keep requesting for an output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}^{(P_{\mathsf{D}},q)}_{j,k}$, until an output is received.

- Add the edge (P_j, P_k) to $G_q^{(i)}$ if $(P_j, \mathsf{Acast}, \mathsf{sid}_{j,k}^{(P_\mathsf{D},q)}, \mathsf{OK}_q(j,k))$ and $(P_k, \mathsf{Acast}, \mathsf{sid}_{k,j}^{(P_\mathsf{D},q)}, \mathsf{OK}_q(k,j))$ is received from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{j,k}^{(P_\mathsf{D},q)}$ and $\mathsf{sid}_{k,j}^{(P_\mathsf{D},q)}$ respectively.

– **Identification of Core Sets and Public Announcements**: P_D executes the following steps to compute the core sets.
  - For each $S_q \in \mathbb{S}$, check if there exists a subset of parties $\mathcal{W}_q \subseteq S_q$, such that $S_q \setminus \mathcal{W}_q \in \mathcal{Z}$, and the parties in $\mathcal{W}_q$ form a clique in the consistency graph G_q^D. If such a $\mathcal{W}_q$ exists, then assign $\mathcal{C}_q := \mathcal{W}_q$.[a]
  - Once $\mathcal{C}_1, \ldots, \mathcal{C}_h$ are computed, send $(\mathsf{sender}, \mathsf{Acast}, \mathsf{sid}_{P_\mathsf{D}}, \{\mathcal{C}_q\}_{S_q \in \mathbb{S}})$ to $\mathcal{F}_{\mathsf{Acast}}$, where $\mathsf{sid}_{P_\mathsf{D}} = \mathsf{sid}||P_\mathsf{D}$.

– **Share computation**: Each $P_i \in \mathcal{P}$ executes the following steps.
  - Keep requesting for output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{P_\mathsf{D}}$ until an output is received.
  - Upon receiving an output $(\mathsf{sender}, \mathsf{Acast}, \mathsf{sid}_{P_\mathsf{D}}, \{\mathcal{C}_q\}_{S_q \in \mathbb{S}})$ from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{P_\mathsf{D}}$, wait until the parties in $\mathcal{C}_q$ form a clique in $G_q^{(i)}$, corresponding to each $S_q \in \mathbb{S}$. For $q = 1, \ldots, h$, verify if $S_q \setminus \mathcal{C}_q \in \mathcal{Z}$. If the verification is successful, then proceed to compute the shares corresponding to each S_q such that $P_i \in S_q$ as follows.

    1. If $P_i \in \mathcal{C}_q$ then set $[s]_q = s_{qi}$ and corresponding to every signer $P_j \in \mathcal{C}_q$, reveal $\mathsf{ICSig}(\mathsf{sid}_{j,i,k}^{(P_\mathsf{D},q)}, P_j, P_i, P_k, s_{qi})$ to every receiver party $P_k \in S_q \setminus \mathcal{C}_q$.
    2. If $P_i \notin \mathcal{C}_q$, then wait till P_i finds some $P_j \in \mathcal{C}_q$ such that P_i has accepted $\mathsf{ICSig}(\mathsf{sid}_{k,j,i}^{(P_\mathsf{D},q)}, P_k, P_j, P_i, s_{qj})$ revealed by the intermediary P_j, corresponding to every signer $P_k \in \mathcal{C}_q$. Then set $[s]_q = s_{qj}$.

  - Upon computing $\{[s]_q\}_{P_i \in S_q}$, output $(\mathsf{share}, \mathsf{sid}, P_\mathsf{D}, \{[s]_q\}_{P_i \in S_q})$.

[a] Similar to the protocol Π_{PVSS}, the existence of the core sets can be verified with $\mathcal{O}(\text{poly}(n, |\mathcal{Z}|))$ computational effort (see the discussion on the computational complexity of the protocol Π_{PVSS}).

Figure 18: The statistically-secure VSS protocol for session id sid for realizing $\mathcal{F}_{\mathsf{VSS}}$ in the $\mathcal{F}_{\mathsf{Acast}}$-hybrid model

Remark 4.8. We stress that similar to the protocol Π_{PVSS}, the honest parties *may not* get any output in the protocol Π_{SVSS}, if P_D *does not* make public valid core sets. On the other hand, if P_D is *honest*, then it will eventually get and broadcast valid core sets, since the set of honest parties will eventually satisfy all the required conditions of valid core sets.

We next prove the properties of the protocol Π_{SVSS}, stated in Theorem 4.9.

Theorem 4.9. *Let $\mathcal{Z}$ satisfy the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. Then Π_{SVSS} UC-securely realizes $\mathcal{F}_{\mathsf{VSS}}$ in the $\mathcal{F}_{\mathsf{Acast}}$-hybrid model, except with probability $|\mathcal{Z}| \cdot n^3 \cdot \epsilon_{\mathsf{AICP}}$, where $\epsilon_{\mathsf{AICP}} \approx \frac{n^2}{|\mathbb{F}|}$. The protocol makes $\mathcal{O}(|\mathcal{Z}| \cdot n^3)$ calls to $\mathcal{F}_{\mathsf{Acast}}$ with $\mathcal{O}(n \cdot \log |\mathbb{F}|)$ bit messages and additionally incurs a communication of $\mathcal{O}(|\mathcal{Z}| \cdot n^4 \log |\mathbb{F}|)$ bits.*

By replacing the calls to $\mathcal{F}_{\mathsf{Acast}}$ with protocol Π_{Acast}, the protocol incurs a total communication of $\mathcal{O}(|\mathcal{Z}| \cdot n^6 \log |\mathbb{F}|)$ bits.

Proof. In the protocol, the dealer needs to send the share s_q to all the parties in S_q, and this requires communication of $\mathcal{O}(|\mathcal{Z}| \cdot n \log |\mathbb{F}|)$ bits. An instance of Π_{Auth} and Π_{Reveal} is executed with respect to every ordered triplet of parties $P_i, P_j, P_k \in S_q$, leading to $\mathcal{O}(|\mathcal{Z}| \cdot n^3)$ instances of Π_{Auth} and Π_{Reveal} being executed. The communication complexity now follows from the communication complexity of Π_{Auth} and Π_{Reveal} (Claim 4.5) and from the communication complexity of the protocol Π_{Acast}.

We next prove the security of the protocol. Let Adv be an arbitrary adversary, attacking the protocol Π_{SVSS} by corrupting a set of parties $Z^\star \in \mathcal{Z}$, and let Env be an arbitrary environment. We show the existence of a simulator $\mathcal{S}_{\mathsf{SVSS}}$, such that for any $Z^\star \in \mathcal{Z}$, the outputs of the honest parties and the view of the adversary in the protocol Π_{SVSS} is indistinguishable from the outputs of the honest parties and the view of the adversary in an execution in the ideal world involving $\mathcal{S}_{\mathsf{SVSS}}$ and $\mathcal{F}_{\mathsf{VSS}}$, except with probability at most $|\mathcal{Z}| \cdot n^3 \cdot \epsilon_{\mathsf{AICP}}$, where $\epsilon_{\mathsf{AICP}} \approx \frac{n^2}{|\mathbb{F}|}$ (see Lemma 4.6). The simulator is very similar to the simulator $\mathcal{S}_{\mathsf{PVSS}}$ for the protocol Π_{PVSS} (see Fig 7), except that the simulator now has to simulate giving and accepting signatures on the behalf of honest parties, as part of pairwise consistency checks. In addition, for each $S_q \in \mathbb{S}$, the simulator has to simulate revealing signatures to the corrupt parties in $S_q \setminus \mathcal{C}_q$ on the behalf of the honest parties in $\mathcal{C}_q$. The simulator is formally presented in Figure 19.

Simulator $\mathcal{S}_{\mathsf{SVSS}}$

$\mathcal{S}_{\mathsf{SVSS}}$ constructs virtual real-world honest parties and invokes the real-world adversary Adv. The simulator simulates the view of Adv, namely its communication with Env, the messages sent by the honest parties and the interaction with $\mathcal{F}_{\mathsf{Acast}}$. In order to simulate Env, the simulator $\mathcal{S}_{\mathsf{PVSS}}$ forwards every message it receives from Env to Adv and vice-versa. The simulator then simulates the various phases of the protocol as follows, depending upon whether the dealer is honest or corrupt.

Simulation When P_{D} is Honest

Interaction with $\mathcal{F}_{\mathsf{VSS}}$: the simulator interacts with the functionality $\mathcal{F}_{\mathsf{VSS}}$ and receives a request based delayed output $(\mathsf{share}, \mathsf{sid}, P_{\mathsf{D}}, \{[s]_q\}_{S_q \cap Z^\star \neq \emptyset})$, on the behalf of the parties in $Z^\star$.

Distribution of Shares: On the behalf of the dealer, the simulator sends $(\mathsf{dist}, \mathsf{sid}, P_{\mathsf{D}}, q, [s]_q)$ to Adv, corresponding to every $P_i \in Z^\star \cap S_q$.

Pairwise Consistency Tests on IC-Signed Values:

- For each $S_q \in \mathbb{S}$ such that $S_q \cap Z^\star \neq \emptyset$, corresponding to each $P_i \in S_q \cap Z^\star$, the simulator does the following.
  - On the behalf of every party $P_j \in S_q \setminus Z^\star$ as a signer and every $P_k \in S_q$ as a receiver, perform the role of the signer and the honest verifiers as per the steps of Π_{Auth} and interact with Adv on the behalf of the honest parties to give $\mathsf{ICSig}(\mathsf{sid}^{(P_{\mathsf{D}},q)}_{j,i,k}, P_j, P_i, P_k, s_{qj})$ to P_i, where $s_{qj} = [s]_q$.
  - On the behalf of every $P_j, P_k \in S_q$ as intermediary and receiver respectively, perform the role of the honest parties as per the steps of Π_{Auth} and interact with Adv on the behalf of the honest parties, if Adv gives the signature $\mathsf{ICSig}(\mathsf{sid}^{(P_{\mathsf{D}},q)}_{i,j,k}, P_i, P_j, P_k, s_{qi})$ to P_j on the behalf of the signer P_i. Upon receiving the signature $\mathsf{ICSig}(\mathsf{sid}^{(P_{\mathsf{D}},q)}_{i,j,k}, P_i, P_j, P_k, s_{qi})$ from P_i, record it.
- For each $S_q \in \mathbb{S}$ and for every $P_i, P_j \in S_q \setminus Z^\star$ the simulator simulates P_i giving $\mathsf{ICSig}(\mathsf{sid}^{(P_{\mathsf{D}},q)}_{i,j,k}, P_i, P_j, P_k, v)$ to P_j, corresponding to each $P_k \in S_q$, by playing the role of the honest parties and interacting with Adv on their behalf, as per the steps of Π_{Auth}, in the respective Π_{Auth} instances. Based on the following conditions, the simulator chooses the value v in these instances as follows.
  - $S_q \cap Z^\star \neq \emptyset$: Choose v to be $[s]_q$.
  - $S_q \cap Z^\star = \emptyset$: Pick a random element from $\mathbb{F}$ as v.

Announcing Results of Pairwise Consistency Tests:

- If for any $S_q \in \mathbb{S}$, Adv requests an output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}^{(P_{\mathsf{D}},q)}_{i,j}$ corresponding to parties $P_i \in S_q \setminus Z^\star$ and $P_j \in S_q$, then the simulator provides the output on the behalf of $\mathcal{F}_{\mathsf{Acast}}$ as follows.
  - If $P_j \in S_q \setminus Z^\star$, then send the output $(P_i, \mathsf{Acast}, \mathsf{sid}^{(P_{\mathsf{D}},q)}_{i,j}, \mathsf{OK}_q(i,j))$.
  - If $P_j \in (S_q \cap Z^\star)$, then send the output $(P_i, \mathsf{Acast}, \mathsf{sid}^{(P_{\mathsf{D}},q)}_{i,j}, \mathsf{OK}_q(i,j))$, if $\mathsf{ICSig}(\mathsf{sid}^{(P_{\mathsf{D}},q)}_{j,i,k}, P_j, P_i, P_k, s_{qj})$ has been recorded on the behalf of P_j as a signer, corresponding to the intermediary P_i and every $P_k \in S_q$ as a receiver, such that $s_{qj} = [s]_q$ holds.
- If for any $S_q \in \mathbb{S}$ and any $P_i \in S_q \cap Z^\star$, Adv sends $(P_i, \mathsf{Acast}, \mathsf{sid}^{(P_{\mathsf{D}},q)}_{i,j}, \mathsf{OK}_q(i,j))$ to $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}^{(P_{\mathsf{D}},q)}_{i,j}$ on the behalf of P_i for any $P_j \in S_q$, then the simulator records it. Moreover, if Adv requests an output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}^{(P_{\mathsf{D}},q)}_{i,j}$, then the simulator sends the output $(P_i, \mathsf{Acast}, \mathsf{sid}^{(P_{\mathsf{D}},q)}_{i,j}, \mathsf{OK}_q(i,j))$ on the behalf of $\mathcal{F}_{\mathsf{Acast}}$.

Construction of Core Sets and Public Announcement:

- For each $S_q \in \mathbb{S}$, the simulator plays the role of P_{D} and adds the edge (P_i, P_j) to the graph $G^{(\mathsf{D})}_q$ over the vertex set S_q, if any one of the following is true.
  1. $P_i, P_j \in S_q \setminus Z^\star$.
  2. If $P_i \in S_q \cap Z^\star$ and $P_j \in S_q \setminus Z^\star$, then the simulator has recorded $(P_i, \mathsf{Acast}, \mathsf{sid}^{(P_{\mathsf{D}},q)}_{i,j}, \mathsf{OK}_q(i,j))$ sent by Adv on the behalf of P_i to $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}^{(P_{\mathsf{D}},q)}_{i,j}$, and has recorded $\mathsf{ICSig}(\mathsf{sid}^{(P_{\mathsf{D}},q)}_{i,j,k}, P_i, P_j, P_k, s_{qi})$

on the behalf of P_i as a signer and P_j as an intermediary corresponding to every party $P_k \in S_q$ as a receiver, such that $s_{qi} = [s]_q$ holds.

3. If $P_i, P_j \in S_q \cap Z^\star$, then the simulator has recorded $(P_i, \mathsf{Acast}, \mathsf{sid}^{(q)}_{i,j}, \mathsf{OK}_q(i,j))$ and $(P_j, \mathsf{Acast}, \mathsf{sid}^{(q)}_{j,i},$ $, \mathsf{OK}_q(j,i))$ sent by Adv on behalf P_i and P_j respectively, to $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}^{(P_\mathsf{D},q)}_{i,j}$ and $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}^{(P_\mathsf{D},q)}_{j,i}$.

– For each $S_q \in \mathbb{S}$, the simulator finds a set $\mathcal{C}_q$ which forms a clique in G^{D}_q, such that $S_q \setminus \mathcal{C}_q \in \mathcal{Z}$. When Adv requests output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{P_\mathsf{D}}$, the simulator sends the output $(\mathsf{sender}, \mathsf{Acast}, \mathsf{sid}_{P_\mathsf{D}}, \{\mathcal{C}_q\}_{S_q \in \mathbb{S}})$ on the behalf of $\mathcal{F}_{\mathsf{Acast}}$.

Share Computation: Once $\mathcal{C}_1, \ldots, \mathcal{C}_q$ are computed, then for each $S_q \in \mathbb{S}$, simulator does the following for every $P_i \in (S_q \setminus \mathcal{C}_q) \cap Z^\star$ and every $P_j \in \mathcal{C}_q \setminus Z^\star$.

– Simulate the revelation of the signature $\mathsf{ICSig}(\mathsf{sid}^{(P_\mathsf{D},q)}_{k,j,i}, P_k, P_j, P_i, s_{qk})$ to P_i on the behalf of the intermediary P_j corresponding to every signer $P_k \in \mathcal{C}_q$, where $s_{qk} = [s]_q$, by playing the role of the honest parties as per Π_{Reveal} and interacting with Adv.

Simulation When P_D is Corrupt

In this case, the simulator $\mathcal{S}_{\mathsf{SVSS}}$ interacts with Adv during the various phases of Π_{SVSS} as follows.

Distribution of Shares: For $q = 1, \ldots, h$, if Adv sends $(\mathsf{dist}, \mathsf{sid}, P_\mathsf{D}, q, v)$ on the behalf of P_D to any party $P_i \in S_q \setminus Z^\star$, then the simulator records it and sets s_{qi} to v.

Pairwise Consistency Tests on IC-Signed Values:

– For each $S_q \in \mathbb{S}$ such that $S_q \cap Z^\star \neq \emptyset$, corresponding to each party $P_i \in S_q \cap Z^\star$ and each $P_j \in S_q \setminus Z^\star$, the simulator does the following.
  - If s_{qj} has been set to some value, then simulate giving $\mathsf{ICSig}(\mathsf{sid}^{(P_\mathsf{D},q)}_{j,i,k}, P_j, P_i, P_k, s_{qj})$ to Adv on the behalf of P_j as a signer, corresponding to every $P_k \in \mathcal{P}$ as receiver, by playing the role of the honest parties as per the steps of Π_{Auth}.
  - Upon receiving $\mathsf{ICSig}(\mathsf{sid}^{(P_\mathsf{D},q)}_{i,j,k}, P_i, P_j, P_k, s_{qi})$ from Adv on the behalf of P_i as a signer, corresponding to $P_j \in S_q$ as an intermediary and $P_k \in S_q$ as a receiver, record $\mathsf{ICSig}(\mathsf{sid}^{(P_\mathsf{D},q)}_{i,j,k}, P_i, P_j, P_k, s_{qi})$.

– For each $S_q \in \mathbb{S}$, corresponding to each party $P_i, P_j \in S_q \setminus Z^\star$, the simulator does the following.
  - Upon setting s_{qi} to some value, simulate P_i giving $\mathsf{ICSig}(\mathsf{sid}^{(P_\mathsf{D},q)}_{i,j,k}, P_i, P_j, P_k, s_{qi})$ to P_j, corresponding to every receiver $P_k \in S_q$, by playing the role of the honest parties and interacting with Adv as per the steps of Π_{Auth}.

Announcing Results of Pairwise Consistency Tests:

– If for any $S_q \in \mathbb{S}$, Adv requests an output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}^{(P_\mathsf{D},q)}_{i,j}$ corresponding to parties $P_i \in S_q \setminus Z^\star$ and $P_j \in S_q$, then the simulator provides the output on the behalf of $\mathcal{F}_{\mathsf{Acast}}$ as follows, if s_{qi} has been set to some value.
  - If $P_j \in S_q \setminus Z^\star$, then send the output $(P_i, \mathsf{Acast}, \mathsf{sid}^{(P_\mathsf{D},q)}_{i,j}, \mathsf{OK}_q(i,j))$, if s_{qj} has been set to some value and $s_{qi} = s_{qj}$ holds.
  - If $P_j \in S_q \cap Z^\star$, then send the output $(P_i, \mathsf{Acast}, \mathsf{sid}^{(P_\mathsf{D},q)}_{i,j}, \mathsf{OK}_q(i,j))$, if $\mathsf{ICSig}(\mathsf{sid}^{(P_\mathsf{D},q)}_{j,i,k}, P_j, P_i, P_k, s_{qj})$ has been recorded on the behalf of P_j as a signer for the intermediary P_i, corresponding to every $P_k \in S_q$ as a receiver, such that $s_{qj} = s_{qi}$ holds.

– If for any $S_q \in \mathbb{S}$ and any $P_i \in S_q \cap Z^\star$, Adv sends $(P_i, \mathsf{Acast}, \mathsf{sid}^{(P_\mathsf{D},q)}_{i,j}, \mathsf{OK}_q(i,j))$ to $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}^{(P_\mathsf{D},q)}_{i,j}$ on the behalf of P_i for any $P_j \in S_q$, then the simulator records it. Moreover, if Adv requests for an output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}^{(P_\mathsf{D},q)}_{i,j}$, then the simulator sends the output $(P_i, \mathsf{Acast}, \mathsf{sid}^{(P_\mathsf{D},q)}_{i,j}, \mathsf{OK}_q(i,j))$ on the behalf of $\mathcal{F}_{\mathsf{Acast}}$.

Construction of Core Sets: For each $S_q \in \mathbb{S}$, the simulator plays the role of the honest parties $P_i \in S_q \setminus Z^\star$ and adds the edge (P_j, P_k) to the graph $G^{(i)}_q$ over vertex set S_q, if any one of the following is true.

- If $P_j, P_k \in S_q \setminus Z^\star$, then the simulator has set s_{qj} and s_{qk} to some values, such that $s_{qj} = s_{qk}$ holds.
- If $P_j \in S_q \cap Z^\star$ and $P_k \in S_q \setminus Z^\star$, then all the following should hold.
  – The simulator has recorded $(P_j, \mathsf{Acast}, \mathsf{sid}^{(P_\mathsf{D},q)}_{j,k}, \mathsf{OK}_q(j,k))$ sent by Adv on the behalf of P_j to $\mathcal{F}_{\mathsf{Acast}}$

with $\mathsf{sid}_{j,k}^{(P_\mathsf{D},q)}$;

– The simulator has recorded $\mathsf{ICSig}(\mathsf{sid}_{j,k,m}^{(P_\mathsf{D},q)}, P_j, P_k, P_m, s_{qj})$ on the behalf of P_j as a signer and P_k as an intermediary, corresponding to every receiver $P_m \in S_q$;

– The simulator has set s_{qk} to a value such that $s_{qj} = s_{qk}$ holds.

- If $P_j, P_k \in S_q \cap Z^\star$, then the simulator has recorded $(P_j, \mathsf{Acast}, \mathsf{sid}_{j,k}^{(P_\mathsf{D},q)}, \mathsf{OK}_q(j,k))$ and $(P_k, \mathsf{Acast}, \mathsf{sid}_{k,j}^{(P_\mathsf{D},q)}, \mathsf{OK}_q(k,j))$ sent by Adv on behalf of P_j and P_k respectively, to $\mathcal{F}_\mathsf{Acast}$ with $\mathsf{sid}_{j,k}^{(P_\mathsf{D},q)}$ and $\mathcal{F}_\mathsf{Acast}$ with $\mathsf{sid}_{k,j}^{(P_\mathsf{D},q)}$.

Verification of Core Sets and Interaction with $\mathcal{F}_\mathsf{VSS}$:

- If Adv sends $(\mathsf{sender}, \mathsf{Acast}, \mathsf{sid}_{P_\mathsf{D}}, \{\mathcal{C}_q\}_{S_q \in \mathbb{S}})$ to $\mathcal{F}_\mathsf{Acast}$ with $\mathsf{sid}_{P_\mathsf{D}}$ on the behalf of P_D, then the simulator records it. Moreover, if Adv requests for an output from $\mathcal{F}_\mathsf{Acast}$ with $\mathsf{sid}_{P_\mathsf{D}}$, then on the behalf of $\mathcal{F}_\mathsf{Acast}$, the simulator sends the output $(P_\mathsf{D}, \mathsf{Acast}, \mathsf{sid}_{P_\mathsf{D}}, \{\mathcal{C}_q\}_{S_q \in \mathbb{S}})$.
- If simulator has recorded the sets $\{\mathcal{C}_q\}_{S_q \in \mathbb{S}}$, then it plays the role of the honest parties and verifies if for $q = 1, \ldots, h$, the set $\mathcal{C}_q$ is valid with respect to S_q, by checking if $S_q \setminus \mathcal{C}_q \in \mathcal{Z}$ and if $\mathcal{C}_q$ constitutes a clique in the graph $G_q^{(i)}$ of every party $P_i \in \mathcal{P} \setminus Z^\star$. If $\mathcal{C}_1, \ldots, \mathcal{C}_q$ are valid, then the simulator sends $(\mathsf{share}, \mathsf{sid}, P_\mathsf{D}, \{s_q\}_{S_q \in \mathbb{S}})$ to $\mathcal{F}_\mathsf{VSS}$, where s_q is set to s_{qi} corresponding to any $P_i \in \mathcal{C}_q \setminus Z^\star$.

Figure 19: Simulator for the protocol Π_SVSS where Adv corrupts the parties in set $Z^\star \in \mathcal{Z}$

We now prove a series of claims, which helps us to prove the theorem. We start with an *honest* P_D.

Claim 4.10. If P_D is honest, then the view of Adv in the simulated execution of Π_SVSS with $\mathcal{S}_\mathsf{PVSS}$ is identically distributed to the view of Adv in the real execution of Π_SVSS involving honest parties.

Proof. Let $\mathbb{S}^\star \stackrel{def}{=} \{S_q \in \mathbb{S} \mid S_q \cap Z^\star \neq \emptyset\}$. Then the view of Adv during the two executions consists of the following.

– **The shares $\{[s]_q\}_{S_q \in \mathbb{S}^\star}$ distributed by P_D**: In the real execution, Adv receives $[s]_q$ from P_D for each $S_q \in \mathbb{S}^\star$. In the simulated execution, the simulator provides this to Adv on behalf of P_D. Clearly, the distribution of the shares is identical in both the executions.

– **Corresponding to every $S_q \in \mathbb{S}^\star$ and every triplet of parties P_i, P_j, P_k where $P_j \in S_q \setminus Z^\star$, $P_i \in S_q \cap Z^\star$ and $P_k \in S_q$, the signature $\mathsf{ICSig}(\mathsf{sid}_{j,i,k}^{(P_\mathsf{D},q)}, P_j, P_i, P_k, s_{qj})$ received from P_j as part of pairwise consistency tests**: While P_j sends this to Adv in the real execution, the simulator sends this on the behalf of P_j in the simulated execution. Clearly, the distribution of the messages learnt by Adv during the corresponding instances of Π_Auth is identical in both the executions.

– **Corresponding to every $S_q \in \mathbb{S}$, every pair of parties $P_i, P_j \in S_q \setminus Z^\star$ and every $P_k \in S_q$, the view generated when P_i gives $\mathsf{ICSig}(\mathsf{sid}_{i,j,k}^{(P_\mathsf{D},q)}, P_i, P_j, P_k, v))$ to P_j**: We consider the following two cases.

  - $S_q \in \mathbb{S}^\star$: In both the real and simulated execution, the value of v is $[s]_q$. Since the simulator simulates the interaction of honest parties with Adv during the simulated execution, the distribution of messages is identical in both the executions.
  - $S_q \notin \mathbb{S}^\star$: In the simulated execution, the simulator chooses v to be a random element from $\mathbb{F}$, while in the real execution, v is $[s]_q$. However, due to the privacy property of AICP (Claim 4.2), the view of Adv is independent of the value of v in either of the executions. Hence, the distribution of the messages is identical in both the executions.

– **For every $S_q \in \mathbb{S}$ and every $P_i, P_j \in S_q$, the outputs $(P_i, \mathsf{Acast}, \mathsf{sid}_{i,j}^{(P_\mathsf{D},q)}, \mathsf{OK}_q(i,j))$ of the pairwise consistency tests, received from $\mathcal{F}_\mathsf{Acast}$ with $\mathsf{sid}_{i,j}^{(P_\mathsf{D},q)}$**: To compare the distribution of these messages in the two executions, we consider the following cases, considering an arbitrary $S_q \in \mathbb{S}$ and arbitrary $P_i, P_j \in S_q$.

- $P_i, P_j \in S_q \setminus Z^\star$: In both the executions, Adv receives $(P_i, \mathsf{Acast}, \mathsf{sid}_{ij}^{(P_\mathsf{D},q)}, \mathsf{OK}_q(i,j))$ as the output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{i,j}^{(P_\mathsf{D},q)}$.
- $P_i \in S_q \setminus Z^\star, P_j \in (S_q \cap Z^\star)$: In both the executions, Adv receives $(P_i, \mathsf{Acast}, \mathsf{sid}_{i,j}^{(P_\mathsf{D},q)}, \mathsf{OK}_q(i,j))$ as the output from $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{i,j}^{(P_\mathsf{D},q)}$ if and only if Adv gave $\mathsf{ICSig}(\mathsf{sid}_{j,i,k}^{P_\mathsf{D},q}, P_j, P_i, P_k, s_{qj})$ on the behalf of P_j to P_i, corresponding to every $P_k \in S_q$, such that $s_{qj} = [s]_q$ holds.
- $P_i \in (S_q \cap Z^\star)$: In both the executions, Adv receives $(P_i, \mathsf{Acast}, \mathsf{sid}_{i,j}^{(q)}, \mathsf{OK}_q(i,j))$ if and only if Adv on the behalf of P_i has sent $(P_i, \mathsf{Acast}, \mathsf{sid}_{i,j}^{(P_\mathsf{D},q)}, \mathsf{OK}_q(i,j))$ to $\mathcal{F}_{\mathsf{Acast}}$ with $\mathsf{sid}_{i,j}^{(P_\mathsf{D},q)}$ for P_j.

Clearly, irrespective of the case, the distribution of the OK messages is identical in both the executions.

- **The core sets** $\{\mathcal{C}_q\}_{S_q \in \mathbb{S}}$: In both the executions, the sets $\mathcal{C}_q$ are determined based on the OK_q messages delivered to P_D. So the distribution of these sets is also identical.

– **Corresponding to every $S_q \in \mathbb{S}^\star$, for every triplet of parties P_i, P_j, P_k where $P_i \in \mathcal{C}_q \setminus Z^\star$, $P_j \in (S_q \setminus \mathcal{C}_q) \cap Z^\star$ and $P_k \in \mathcal{C}_q$, the signatures $\mathsf{ICSig}(\mathsf{sid}_{k,i,j}^{P_\mathsf{D},q}, P_k, P_i, P_j, s_{qk})$ revealed by party P_i to P_j, signed by party P_k**: We note that the distribution of core sets $\mathcal{C}_q$ is the same in both the executions. In the real execution, P_i, upon receiving $\mathsf{ICSig}(\mathsf{sid}_{k,i,j}^{P_\mathsf{D},q}, P_k, P_i, P_j, s_{qk})$ during Π_{Auth}, checks if $s_{qk} = s_{qi}$ holds, before adding the edge (P_i, P_k) in G_q^i. Since P_D is honest, $s_{qi} = [s]_q$. In the simulated execution as well, the simulator reveals $\mathsf{ICSig}(\mathsf{sid}_{k,i,j}^{P_\mathsf{D},q}, P_k, P_i, P_j, s_{qk})$ to Adv, where $s_{qk} = [s]_q$. Hence, the distribution of messages is identical in both executions. □

We next claim that if the dealer is *honest*, then conditioned on the view of the adversary Adv (which is identically distributed in both the executions, as per the previous claim), the outputs of the honest parties are identically distributed in both the executions.

Claim 4.11. If P_D is honest, then conditioned on the view of Adv, the output of the honest parties during the execution of Π_{SVSS} involving Adv has the same distribution as the output of the honest parties in the ideal-world involving $\mathcal{S}_{\mathsf{PVSS}}$ and $\mathcal{F}_{\mathsf{VSS}}$, except with probability at most $|\mathcal{Z}| \cdot n^3 \cdot \epsilon_{\mathsf{AICP}}$, where $\epsilon_{\mathsf{AICP}} \approx \frac{n^2}{|\mathbb{F}|}$.

Proof. Let P_D be honest and let View be an arbitrary view of Adv. Moreover, let $\{s_q\}_{S_q \cap Z^\star \neq \emptyset}$ be the shares of the corrupt parties, as per View. Furthermore, let $\{s_q\}_{S_q \cap Z^\star = \emptyset}$ be the shares used by P_D in the simulated execution corresponding to the set $S_q \in \mathbb{S}$, such that $S_q \cap Z^\star = \emptyset$. Let $s \stackrel{def}{=} \sum_{S_q \cap Z^\star \neq \emptyset} s_q + \sum_{S_q \cap Z^\star = \emptyset} s_q$. Then, in the simulated execution, each *honest* party P_i obtains the output $\{[s]_q\}_{P_i \in S_q}$ from $\mathcal{F}_{\mathsf{VSS}}$, where $[s]_q = s_q$. We now show that except with probability at most $|\mathcal{Z}| \cdot n^3 \cdot \epsilon_{\mathsf{AICP}}$, each honest P_i eventually obtains the output $\{[s]_q\}_{P_i \in S_q}$ in the real execution as well, if P_D's inputs in the protocol Π_{SVSS} are $\{s_q\}_{S_q \in \mathbb{S}}$.

Since P_D is *honest*, it sends the share s_q to *all* the parties in the set S_q, which is eventually delivered. Now consider *any* $S_q \in \mathbb{S}$. During the pairwise consistency tests, each *honest* $P_k \in S_q$ will eventually send $\mathsf{ICSig}(\mathsf{sid}_{k,j,m}^{(P_\mathsf{D},q)}, P_k, P_j, P_m, s_{qk})$ to *all* the parties P_j in S_q, with respect to every receiver $P_m \in \mathcal{P}$, where $s_{qk} = s_q$. Consequently, every *honest* $P_j \in S_q$ will eventually broadcast the $\mathsf{OK}_q(j,k)$ message, corresponding to every *honest* $P_k \in S_q$. This is because, by the correctness of AICP (Claim 4.1), P_j will receive s_{qk}, and $s_{qj} = s_{qk} = s_q$ will hold. So, every *honest* party (including P_D) eventually receives the $\mathsf{OK}_q(j,k)$ messages This implies that the parties in $S_q \setminus Z^\star$ will eventually form a clique in the graph $G_q^{(i)}$ of every *honest* P_i. This further implies that P_D will eventually find a set $\mathcal{C}_q$ where $S_q \setminus \mathcal{C}_q \in \mathcal{Z}$ and where $\mathcal{C}_q$ constitutes a clique in the consistency graph of every honest party. This is because the set $S_q \setminus Z^\star$ is guaranteed to eventually constitute a clique. Hence, P_D eventually broadcasts the sets $\{\mathcal{C}_q\}_{S_q \in \mathbb{S}}$, which are eventually delivered to every honest party. Moreover, the verification of these sets will eventually be successful for every honest party.

Next consider an arbitrary S_q and an arbitrary *honest* $P_i \in S_q$. If $P_i \in \mathcal{C}_q$, then it has already received the share s_{qi} from P_{D} and $s_{qi} = s_q$ holds. Hence, P_i sets $[s]_q$ to s_q. So consider the case when $P_i \notin \mathcal{C}_q$. In this case, P_i waits to find some $P_j \in \mathcal{C}_q$ such that P_i accepts the signature $\mathsf{ICSig}(\mathsf{sid}^{(P_{\mathsf{D}},q)}_{k,j,i}, P_k, P_j, P_i, s_{qj})$ from intermediary P_j, corresponding to every signer $P_k \in \mathcal{C}_q$ and upon finding such a P_j, party P_i sets $[s]_q$ to s_{qj}. We show that except with probability at most $n \cdot \epsilon_{\mathsf{AICP}}$, party P_i will eventually find a candidate P_j satisfying the above condition. Moreover, if P_i finds a candidate P_j satisfying the above condition, then except with probability at most $n \cdot \epsilon_{\mathsf{AICP}}$, the condition $s_{qj} = s_q$ holds. As P_i can have up to $\mathcal{O}(n)$ candidates for P_j, it will follow from the union bound that except with probability at most $n^2 \cdot \epsilon_{\mathsf{AICP}}$, party P_i will eventually compute $[s]_q$. Now assuming these statements are true, the proof follows from the union bound and the fact that S_q can be any set out of $|\mathcal{Z}|$ subsets in $\mathbb{S}$ and for any S_q, there could be upto $\mathcal{O}(n)$ honest parties P_i in $S_q \setminus C_q$. We next proceed to prove the above two statements.

Since $\mathbb{S}$ satisfies the $\mathbb{Q}^{(2)}(\mathbb{S}, \mathcal{Z})$ condition and $S_q \setminus C_q \in \mathcal{Z}$, it follows that $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(1)}(\mathcal{C}_q, \mathcal{Z})$ condition and hence $\mathcal{C}_q$ contains at least one *honest* party, say P_h. Consider any arbitrary $P_k \in \mathcal{C}_q$. From the protocol steps, P_h has broadcasted the $\mathsf{OK}_q(h, k)$ after receiving $\mathsf{ICSig}(\mathsf{sid}^{(P_{\mathsf{D}},q)}_{k,h,i}, P_k, P_h, P_i, s_{qk})$ from P_k during Π_{Auth} and verifying that $s_{qk} = s_{qh}$ holds, where $s_{qh} = s_q$. It then follows from Lemma 4.6, that except with probability at most ϵ_{AICP}, party P_i will accept the signature $\mathsf{ICSig}(\mathsf{sid}^{(P_{\mathsf{D}},q)}_{k,h,i}, P_k, P_h, P_i, s_{qh})$ revealed by P_h. Hence, except with probability at most $n \cdot \epsilon_{\mathsf{AICP}}$, party P_i will eventually accept the signature $\mathsf{ICSig}(\mathsf{sid}^{(P_{\mathsf{D}},q)}_{k,h,i}, P_k, P_h, P_i, s_{qh})$ corresponding to *all* $P_k \in \mathcal{C}_q$, revealed by P_h.

Finally, consider an *arbitrary* $P_j \in \mathcal{C}_q$, such that P_i has accepted the signature $\mathsf{ICSig}(\mathsf{sid}^{(P_{\mathsf{D}},q)}_{k,j,i}, P_k, P_j, P_i, s_{qj})$ corresponding to *all* $P_k \in \mathcal{C}_q$ and sets $[s]_q$ to s_{qj}. Now one of these signatures corresponds to the signer $P_k = P_h$. If P_j is *corrupt*, then it follows from Lemma 4.6, that except with probability at most ϵ_{AICP}, the condition $s_{qj} = s_{qh}$ holds. As there can be up to $\mathcal{O}(n)$ honest parties P_h in $\mathcal{C}_q$, it follows that P_j will fail to reveal signature of any honest party from $\mathcal{C}_q$ on any $s_{qj} \neq s_q$, except with probability at most $n \cdot \epsilon_{\mathsf{AICP}}$. Since there can be up to $\mathcal{O}(n)$ corrupt parties $P_j \in \mathcal{C}_q$, it then follows from the union bound that except with error probability $n^2 \cdot \epsilon_{\mathsf{AICP}}$, no corrupt party from $\mathcal{C}_q$ will be able to forge the signature of any honest party from $\mathcal{C}_q$ on an incorrect s_q. □

We next prove certain claims with respect to a *corrupt* dealer. The first claim is that the view of Adv in this case is also identically distributed in both the real as well as simulated execution. This is simply because in this case, the *honest* parties have *no* inputs and the simulator simply plays the role of the honest parties, *exactly* as per the steps of the protocol Π_{SVSS} in the simulated execution.

Claim 4.12. If P_{D} is corrupt, then the view of Adv in the simulated execution of Π_{SVSS} with $\mathcal{S}_{\mathsf{PVSS}}$ is identically distributed to the view of Adv in the real execution of Π_{SVSS} involving honest parties.

Proof. The proof follows from the fact that if P_{D} is *corrupt*, then $\mathcal{S}_{\mathsf{PVSS}}$ participates in a full execution of the protocol Π_{SVSS} by playing the role of the honest parties as per the steps of Π_{SVSS}. Hence, there is a one-to-one correspondence between simulated executions and real executions. □

We finally claim that if the dealer is *corrupt*, then conditioned on the view of the adversary (which is identical in both the executions as per the last claim), the outputs of the honest parties are identically distributed in both the executions.

Claim 4.13. If D is corrupt, then conditioned on the view of Adv, the output of the honest parties during the execution of Π_{SVSS} involving Adv has the same distribution as the output of the honest parties in the ideal-world involving $\mathcal{S}_{\mathsf{PVSS}}$ and $\mathcal{F}_{\mathsf{VSS}}$, except with probability at most $|\mathcal{Z}| \cdot n^3 \cdot \epsilon_{\mathsf{AICP}}$, where $\epsilon_{\mathsf{AICP}} \approx \frac{n^2}{|\mathbb{F}|}$.

Proof. Let P_{D} be *corrupt* and let View be an arbitrary view of Adv. We note that it can be found out from View whether valid core sets $\{\mathcal{C}_q\}_{S_q \in \mathbb{S}}$ have been generated during the corresponding execution of Π_{SVSS} or not. We now consider the following cases.

- *No core sets* $\{\mathcal{C}_q\}_{S_q \in \mathbb{S}}$ *are generated as per* View: In this case, the honest parties do not obtain any output in either execution. This is because in the real execution of Π_{SVSS}, the honest parties compute their output only when they get valid core sets $\{\mathcal{C}_q\}_{S_q \in \mathbb{S}}$ from P_{D}'s broadcast. If this is not the case, then in the simulated execution, the simulator $\mathcal{S}_{\mathsf{PVSS}}$ does not provide any input to $\mathcal{F}_{\mathsf{VSS}}$ on behalf of P_{D}; hence, $\mathcal{F}_{\mathsf{VSS}}$ does not produce any output for the honest parties.
- *Core sets* $\{\mathcal{C}_q\}_{S_q \in \mathbb{S}}$ *generated as per* View *are invalid*: Again, in this case, the honest parties do not obtain any output in either execution. This is because in the real execution of Π_{SVSS}, even if the sets $\{\mathcal{C}_q\}_{S_q \in \mathbb{S}}$ are received from P_{D}'s broadcast, the honest parties compute their output only when each $\mathcal{C}_q$ set is found to be *valid* with respect to the verifications performed by the honest parties in their own consistency graphs. If these verifications fail (implying that the core sets are invalid), then in the simulated execution, the simulator $\mathcal{S}_{\mathsf{PVSS}}$ does not provide any input to $\mathcal{F}_{\mathsf{VSS}}$ on behalf of P_{D}, implying that $\mathcal{F}_{\mathsf{VSS}}$ does not produce any output for the honest parties.
- *Valid core sets* $\{\mathcal{C}_q\}_{S_q \in \mathbb{S}}$ *are generated as per* View: We first note that in this case, P_{D} has distributed some common share, say s_q, as determined by View, to all the parties in $\mathcal{C}_q \setminus Z^\star$, during the real execution of Π_{SVSS}. This is because all the parties in $\mathcal{C}_q \setminus Z^\star$ are *honest*, and form a clique in the consistency graph of the honest parties. Hence, each $P_j, P_k \in \mathcal{C}_q \setminus Z^\star$ has broadcasted the messages $\mathsf{OK}_q(j,k)$ and $\mathsf{OK}_q(k,j)$ after checking that $s_{qj} = s_{qk}$ holds, where s_{qj} and s_{qk} are the values received from P_{D} by P_j and P_k respectively.
  We next show that in the real execution of Π_{SVSS}, except with probability at most $n^3 \cdot \epsilon_{\mathsf{AICP}}$, all *honest* parties in $S_q \setminus Z^\star$ eventually set $[s]_q$ to s_q. While this is obviously true for the parties in $\mathcal{C}_q \setminus Z^\star$, the proof when $P_i \in S_q \setminus (Z^\star \cup \mathcal{C}_q)$ is exactly the *same*, as in Claim 4.11.
  Since $|\mathbb{S}| = |\mathcal{Z}|$, it then follows that in the real execution, except with probability at most $n^3 \cdot \epsilon_{\mathsf{AICP}}$, every honest party P_i eventually outputs $\{[s]_q = s_q\}_{P_i \in S_q}$. From the steps of $\mathcal{S}_{\mathsf{PVSS}}$, the simulator sends the shares $\{s_q\}_{S_q \in \mathbb{S}}$ to $\mathcal{F}_{\mathsf{VSS}}$ on the behalf of P_{D} in the simulated execution. Consequently, in the simulated execution, $\mathcal{F}_{\mathsf{VSS}}$ will eventually deliver the shares $\{[s]_q = s_q\}_{P_i \in S_q}$ to every honest I. Hence, except with probability at most $|\mathcal{Z}| \cdot n^3 \cdot \epsilon_{\mathsf{AICP}}$, the outputs of the honest parties are identical in both the executions.

□

The proof of the theorem now follows from Claims 4.10-4.13. □

4.2.1 Statistically-Secure VSS for Superpolynomial $|\mathcal{Z}|$

The error probability of Π_{SVSS} depends linearly on $|\mathcal{Z}|$ (Theorem 4.9), which is problematic for a large-sized $\mathcal{Z}$. The reason for the error probability being dependent on $|\mathcal{Z}|$ is that the protocol involves $\Omega(|\mathcal{Z}|)$ probabilistic checks and during each of these checks, a party who has behaved maliciously, might remain undetected with probability ϵ_{AICP}. In more detail, in each invocation of $\Pi_{\mathsf{Auth}}/\Pi_{\mathsf{Reveal}}$, a cheating attempt of a malicious party is not detected with probability ϵ_{AICP}. As there are $\Theta(|\mathcal{Z}|)$ invocations of $\Pi_{\mathsf{Auth}}/\Pi_{\mathsf{Reveal}}$ per instance of Π_{SVSS}, the resulting error probability depends linearly on $|\mathcal{Z}|$. To avoid this, we use the idea of *local dispute control* to deal with detected cheaters, which was also used in the *synchronous* statistical VSS protocol of [24] to deal with superpolynomial sized adversary structures. This will ensure that the error probability of Π_{SVSS} is only $n^3 \cdot \epsilon_{\mathsf{AICP}}$, *irrespective* of the size of $\mathcal{Z}$ and the number of invocations of Π_{SVSS}.

In the dispute-control framework, the parties locally discard corrupt parties *the moment* they are caught cheating during *any* instance of $\Pi_{\mathsf{Auth}}/\Pi_{\mathsf{Reveal}}$. And once a party P_j is locally discarded by some honest party P_i, then P_i "behaves" as if P_j has *certainly* behaved maliciously in all "future" instances of

$\Pi_{\mathsf{Auth}}/\Pi_{\mathsf{Reveal}}$, *irrespective* of whether this is the case or not. Consequently, the adversary now will have *only* a "fixed" number of attempts to cheat and the *total* error probability of *arbitrary* many instances of $\Pi_{\mathsf{Auth}}/\Pi_{\mathsf{Reveal}}$ will no longer depend on $|\mathcal{Z}|$. In the sequel, we first discuss the modifications in the AICP to handle the disputes, followed by the modifications needed in the protocol Π_{SVSS}.

Each party P_i now maintains a list of *locally-discarded* parties $\mathcal{LD}^{(i)}$ which it keeps populating across *all* instances of Π_{Auth} and Π_{Reveal}, as soon as P_i identifies any party cheating. The way P_i discards parties, it will be guaranteed that an *honest* P_i *never* discards an *honest* party. We first present the modifications made in the protocol Π_{Auth} to locally discard parties.

Modifications in Π_{Auth}. In any instance of Π_{Auth}, if P_i is present in the corresponding set of supporting verifiers $\mathcal{SV}$ (i.e. $P_i \in \mathcal{SV}$), then P_i includes the corresponding signer P_j of the Π_{Auth} instance to $\mathcal{LD}^{(i)}$, if *both* the following conditions hold.

- P_j broadcasts an OK message during the Π_{Auth} instance;
- The linear combination $dv_i + m_i$ of the verification point (v_i, m_i) of P_i with respect to the linear combiner d, *does not* lie on the masked polynomial $B(x)$, broadcasted by the corresponding intermediary.

The idea here is that if P_i is *honest* and if the above conditions hold during any Π_{Auth} instance, then clearly the corresponding signer P_j is *corrupt*. This is because if P_j was honest, then it should have broadcasted an NOK message, after finding that the point $dv_i + m_i$ *does not* lie on the polynomial $B(x)$.

Once P_i discards P_j, then in any pair of $(\Pi_{\mathsf{Auth}}, \Pi_{\mathsf{Reveal}})$ instance involving the signer P_j, if P_i is present in the corresponding set $\mathcal{SV}$, then in the Π_{Reveal} instance, *instead* of revealing the verification point received from P_j, party P_i reveals a *special* publicly-known "dummy" point to the corresponding receiver.[16] Upon receiving the special dummy point, the receiver *always* accepts it, irrespective of what polynomial is revealed by the corresponding intermediary.

The above modification ensures that if in any instance of Π_{Auth} involving a *corrupt signer* P_j and an *honest* intermediary, if P_j distributes an inconsistent verification point to an *honest verifier* P_i from the corresponding $\mathcal{SV}$ set and *still* broadcasts an OK message during Π_{Auth}, then except with probability ϵ_{AICP}, the signer will be locally discarded by the verifier P_i. From then onward, in *all* the instances of $(\Pi_{\mathsf{Auth}}, \Pi_{\mathsf{Reveal}})$, involving the signer P_j, if the verifier P_i is added to the $\mathcal{SV}$ set, then verification point revealed by the verifier P_i during Π_{Reveal} will *always* be considered as accepted, irrespective of what verification point it actually receives from the signer. We stress that the above modification *does not* help a *corrupt* intermediary to forge an *honest* signer's signature towards an *honest* receiver, with the help of potentially *corrupt* verifiers.

Modifications in Π_{Reveal}. We next discuss the modifications needed in the protocol Π_{Reveal} to handle disputes. Consider an instance of Π_{Reveal}, where P_i is the *receiver*. If P_i is sure that the corresponding *intermediary* P_j has tried to forge an incorrect signature, then P_i adds P_j to $\mathcal{LD}^{(i)}$. From then onward, in any instance of Π_{Reveal} involving P_j as intermediary and P_i as the receiver, P_i *rejects* the IC-signature revealed by P_i, irrespective of what data is revealed by P_j.

To check whether P_j has tried to forge an incorrect signature or not during an instance of Π_{Reveal}, party P_i has to be sure that it has *not accepted* the verification-point of some *honest* verifier belonging to the corresponding set $\mathcal{SV}$. To achieve this goal, P_i now *additionally* checks during Π_{Reveal} if there exists a subset of verifiers $\mathcal{SV}'' \subseteq \mathcal{SV}$, where $\mathcal{SV} \setminus \mathcal{SV}'' \in \mathcal{Z}$, such that the verification-points received from *all* the parties in $\mathcal{SV}''$ are *not accepted*. If such a subset $\mathcal{SV}''$ exists, then clearly the intermediary P_j has cheated, since the subset $\mathcal{SV}''$ is bound to contain at least one honest verifier. And if the verification point of an honest verifier is *not considered* as accepted, then clearly P_j is corrupt.

The above modification ensures that if in any instance of Π_{Reveal} involving an *honest* signer, a *corrupt intermediary* P_j and an *honest receiver* P_i, if P_j tries to reveal an incorrect signature during Π_{Reveal}, then

[16] This dummy point serves as an indicator for the receiver that P_i is in conflict with P_j.

except with probability ϵ_{AICP}, the intermediary P_j will be locally discarded by the receiver P_i. And from then onward, in *all* the instances of $(\Pi_{\mathsf{Auth}}, \Pi_{\mathsf{Reveal}})$, involving P_j as the intermediary and P_i as the receiver, the signature revealed by P_j during Π_{Reveal} will *always* be rejected, irrespective of what data is actually revealed by P_j.

Modifications in Π_{SVSS}. Recall that in our *modified* Π_{Auth} protocol, an *honest* verifier will be able to locally catch and discard a corrupt signer, *only when* the signer broadcasts an OK message during Π_{Auth}, even after distributing an inconsistent verification point to the verifier. Taking this into account, we make the following modifications in the Π_{SVSS} protocol, when parties exchange the signed versions of the supposedly common share during the pairwise-consistency tests, where the parties now invoke instances of the modified Π_{Auth} protocol to verify the consistency of the data distributed by the underlying signer parties. Consider an *arbitrary ordered* pair of parties (P_j, P_i). If $P_j, P_i \in S_q$, then during the pairwise consistency test corresponding to S_q, party P_i broadcasts an $\mathsf{OK}_q(i, j)$ message, *only if* P_i finds *both* the following conditions to be true with respect to P_j.

- The signed value s_{qj} received by P_i from P_j is the same as the value s_{qi} received from the dealer.
- In *all* the instances of Π_{Auth} where P_j plays the role of the *signer* and P_i plays the role of the *intermediary*, party P_j has broadcasted only OK messages.[17]

The rest of the protocol steps of Π_{SVSS} remains as it is. The above modifications in $\Pi_{\mathsf{Auth}}, \Pi_{\mathsf{Reveal}}$, along with the way pairwise consistency tests are now performed in the Π_{SVSS} protocol, it will be ensured that if a *corrupt* signer party P_j in any core set $\mathcal{C}_q$ gives an incorrect verification-point to any *honest* verifier P_i, with respect to any *honest* intermediary $P_k \in \mathcal{C}_q$, during *any* instance of Π_{Auth}, then P_j will be caught and locally discarded by P_i, except with probability ϵ_{AICP}. And from then on, P_j will *not* have any chance of cheating the verifier P_i in any Π_{Auth} instance. By considering all possibilities for a *corrupt* signer and an *honest* verifier and an *honest* intermediary, it follows that except with probability at most $n^3 \cdot \epsilon_{\mathsf{AICP}}$, the verification-points of all *honest* verifiers will be accepted by every *honest* receiver during all the instances of Π_{Reveal} in any instance of Π_{SVSS}. And consequently, except with probability at most $n^3 \cdot \epsilon_{\mathsf{AICP}}$, the signatures revealed by all *honest* intermediary parties will always be accepted.

On the other hand, if any *corrupt* intermediary P_j in any core set $\mathcal{C}_q$ tries to forge the signature of an *honest* signer P_k from $\mathcal{C}_q$ to any *honest* receiver P_i, then except with probability ϵ_{AICP}, P_j will be *discarded* by P_i. And from then on, P_i will always reject any signature revealed by P_j. Hence, by considering all possibilities for a *corrupt* intermediary, *honest* signer and *honest* receiver, it follows that except with probability at most $n^3 \cdot \epsilon_{\mathsf{AICP}}$, no *corrupt* intermediary will be able to forge an honest signer's signature to any *honest* receiver in any instance of Π_{SVSS}.

Based on the above discussion, we state the following lemma.

Lemma 4.14. *The modified Π_{SVSS} has an error probability of $n^3 \cdot \epsilon_{\mathsf{AICP}}$, independent of the number of invocations.*

4.3 Statistically-Secure Multiplication Protocol

We next proceed to design our statistically-secure multiplication protocol with $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition, which will be used in our statistically-secure pre-processing phase protocol. We first start with a *non-robust* opti-

[17]Recall that in the original Π_{SVSS} protocol, apart from checking the pairwise consistency of the signed s_{qj} and s_{qi}, party *does not* check for anything additional, for broadcasting the $\mathsf{OK}_q(i, j)$ message. Specifically, if $P_i, P_j \in S_q$, then while receiving the signatures $\mathsf{ICSig}(\mathsf{sid}^{(P_{\mathsf{D}}, q)}_{j,i,\star}, P_j, P_i, \star, s_{qj})$ from P_j, party P_i as an intermediary, *does not* care whether P_j as a signer has broadcasted OK or NOK during the underlying Π_{Auth} instances. But now in the modified protocol, *along* with S_q, for *every* other subset $S_{q'} \in \mathbb{S}$ where $P_j, P_i \in S_{q'}$, in *all* the instances of Π_{Auth} involving P_j as the signer and P_i as the intermediary, P_i checks whether P_j has issued the required signatures to P_i and that too, by broadcasting OK messages in the underlying Π_{Auth} instances, while giving signatures to P_i.

mistic multiplication protocol.

4.3.1 The Basic Multiplication Protocol

Our starting point is the basic multiplication protocol of [24] in the *synchronous* setting. The protocol takes $[a], [b]$, along with a set of *globally-discarded* parties $\mathcal{GD}$ which are *guaranteed* to be corrupt, and outputs $[c]$. In the protocol, each summand $[a]_p[b]_q$ is assigned to a *publicly-known* designated party from $\mathcal{P} \setminus \mathcal{GD}$. Every designated summand-sharing party then secret-shares the sum of all the assigned summands, based on which the parties compute $[c]$. If no summand-sharing party behaves maliciously, then $c = ab$ holds.

Similar to Π_{OptMult}, the main challenge while running the above protocol in the *asynchronous* setting is that a potentially corrupt summand-sharing party may *never* share the sum of the summands assigned to it. To deal with this issue, similar to what was done for Π_{OptMult}, we ask *each* party in $\mathcal{P} \setminus \mathcal{GD}$ to share the sum of all possible summands it is capable of, while ensuring that no summand is shared twice. The idea here is that since $\mathcal{Z}$ now satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition, for every summand $[a]_p[b]_q$, the set $(S_p \cap S_q) \setminus \mathcal{GD}$ is guaranteed to contain at least one *honest* party who will share $[a]_p[b]_q$. Based on this above idea, we design a protocol $\Pi_{\mathsf{BasicMult}}$ which is executed with respect to a set $\mathcal{GD}$, and an iteration number iter. The protocol is almost the same as the protocol Π_{OptMult}, except that it *does not* take any subset $Z \in \mathcal{Z}$ as input.[18] Consequently, the various dynamic sets and session ids maintained in the protocol *will* not be annotated with Z (unlike the protocol Π_{OptMult}).

The protocol proceeds similar to how Π_{OptMult} (Fig 10) does for the perfect setting. In the protocol, each party in $\mathcal{P} \setminus \mathcal{GD}$ shares the sum of all the summands it is capable of sharing. The protocol proceeds in "hops", and a distinct summand-sharing party is selected in each hop. While voting for a candidate summand-sharing party, the parties ensure that the candidate has indeed secret shared *some* sum, that it was not selected in an *earlier* hop, and this it is *not* a part of $\mathcal{GD}$ (see Fig 20 for the formal details).

Protocol $\Pi_{\mathsf{BasicMult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], \mathcal{GD}, \mathsf{iter})$

- **Initialization**:
  - $\mathsf{SIS}_{\mathsf{iter}} = \{(p, q)\}_{p,q=1,\ldots,|\mathbb{S}|}$;
  - $\mathsf{Selected}_{\mathsf{iter}} = \emptyset$;
  - $\mathsf{hop} = 1$;
  - Corresponding to each $P_j \in \mathcal{P} \setminus \mathcal{GD}$, $\mathsf{SIS}^{(j)}_{\mathsf{iter}} = \{(p, q)\}_{P_j \in S_p \cap S_q}$.
- While $\mathsf{SIS}_{\mathsf{iter}} \neq \emptyset$, do the following:
  - **Sharing Summands**: Same as in Π_{OptMult}, except that P_i randomly secret-shares $c^{(i)}_{\mathsf{iter}} = \sum_{(p,q) \in \mathsf{SIS}^{(i)}_{\mathsf{iter}}} [a]_p[b]_q$ by calling $\mathcal{F}_{\mathsf{VSS}}$ with $\mathsf{sid}_{\mathsf{hop},i} \overset{def}{=} \mathsf{sid}||\mathsf{hop}||i$, provided $P_i \notin \mathsf{Selected}_{\mathsf{iter}}$.
  - **Selecting Summand-Sharing Party Through ACS**: Same as in Π_{OptMult}, except that $(\mathsf{vote}, \mathsf{sid}_{\mathsf{hop},j}, 1)$ is sent to $\mathcal{F}_{\mathsf{ABA}}$ with $\mathsf{sid}_{\mathsf{hop},j}$ corresponding to any $P_j \in \mathcal{P}$, if all the following hold:
    - $P_j \notin \mathcal{GD}$;
    - $P_j \notin \mathsf{Selected}_{\mathsf{iter}}$; and
    - An output $(\mathsf{share}, \mathsf{sid}_{\mathsf{hop},j}, P_j, \{[c^{(j)}_{\mathsf{iter}}]_q\}_{P_i \in S_q})$ is received from $\mathcal{F}_{\mathsf{VSS}}$ with $\mathsf{sid}_{\mathsf{hop},j}$, corresponding to the dealer P_j.

    If P_j is selected as common summand-sharing party for this hop, then update the following.
    - $\mathsf{Selected}_{\mathsf{iter}} = \mathsf{Selected}_{\mathsf{iter}} \cup \{P_j\}$.
    - $\mathsf{SIS}_{\mathsf{iter}} = \mathsf{SIS}_{\mathsf{iter}} \setminus \mathsf{SIS}^{(j)}_{\mathsf{iter}}$.
    - $\forall P_k \in \mathcal{P} \setminus \{\mathcal{GD} \cup \mathsf{Selected}_{\mathsf{iter}}\}$: $\mathsf{SIS}^{(k)}_{\mathsf{iter}} = \mathsf{SIS}^{(k)}_{\mathsf{iter}} \setminus \mathsf{SIS}^{(j)}_{\mathsf{iter}}$.
    - $\mathsf{hop} = \mathsf{hop} + 1$.

[18] This is because there *may not* be any *honest* party in the set $(S_p \cap S_q) \setminus Z$, possessing the shares $[a]_p$ as well as $[b]_q$, who can secret-share the summand $[a]_p[b]_q$, since we are *now* working with the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition.

– For every $P_j \in \mathcal{P} \setminus \mathsf{Selected}_{\mathsf{iter}}$, participate in an instance of Π_{PerDefSh} with public input $c^{(j)} = 0$.

– **Output**: Output $\{[c^{(1)}_{\mathsf{iter}}]_q, \ldots, [c^{(n)}_{\mathsf{iter}}]_q, [c_{\mathsf{iter}}]_q\}_{P_i \in S_q}$, where $c_{\mathsf{iter}} \stackrel{def}{=} c^{(1)}_{\mathsf{iter}} + \ldots + c^{(n)}_{\mathsf{iter}}$.

Figure 20: Non-robust basic multiplication protocol in the $(\mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{ABA}})$-hybrid model for session id sid. The above code is executed by every party P_i

We next formally prove the properties of the protocol $\Pi_{\mathsf{BasicMult}}$. While proving these properties, we will assume that $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. This further implies that the sharing specification $\mathbb{S} = (S_1, \ldots, S_h) \stackrel{def}{=} \{\mathcal{P} \setminus Z | Z \in \mathcal{Z}\}$ satisfies the $\mathbb{Q}^{(2)}(\mathbb{S}, \mathcal{Z})$ condition. Moreover, while proving these properties, we assume that no honest party is ever included in the set $\mathcal{GD}$. Looking ahead, the latter condition will be ensured in our next protocol $\Pi_{\mathsf{RandMultCI}}$, where $\Pi_{\mathsf{BasicMult}}$ is used as a subprotocol and where the set $\mathcal{GD}$ is maintained. We first show that the intersection of any two sets in $\mathbb{S}$ contains at least one honest party *outside* $\mathcal{GD}$.

Claim 4.15. For every $Z \in \mathcal{Z}$ and every ordered pair $(p, q) \in \{1, \ldots, h\} \times \{1, \ldots, h\}$, the set $(S_p \cap S_q) \setminus \mathcal{GD}$ contains at least one honest party.

Proof. From the definition of the sharing specification $\mathbb{S}$, we have $S_p = \mathcal{P} \setminus Z_p$ and $S_q = \mathcal{P} \setminus Z_q$, where $Z_p, Z_q \in \mathcal{Z}$. Let $Z^\star \in \mathcal{Z}$ be the set of corrupt parties during the protocol $\Pi_{\mathsf{BasicMult}}$. Now, $S_p \cap S_q = (\mathcal{P} \setminus Z_p) \cap (\mathcal{P} \setminus Z_q) = \mathcal{P} \setminus (Z_p \cup Z_q)$. This means that $(S_p \cap S_q) \cup Z_p \cup Z_q = \mathcal{P}$. If $(S_p \cap S_q) \subseteq Z^\star$, then $\mathcal{P} \subseteq Z^\star \cup Z_p \cup Z_q$. This is a violation of the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition, and hence, $S_p \cap S_q$ contains at least one honest party. Since $\mathcal{GD}$ contains only corrupt parties, $(S_p \cap S_q) \setminus \mathcal{GD}$ contains at least one honest party. □

We next claim a series of properties related to protocol $\Pi_{\mathsf{BasicMult}}$ whose proofs are almost identical to the proof of the corresponding properties for protocol Π_{OptMult}. Hence, we skip the formal proofs.

Claim 4.16. For any iter, if all honest parties participate during the hop number hop in the protocol $\Pi_{\mathsf{BasicMult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], \mathsf{iter})$, then all honest parties eventually obtain a common summand-sharing party, say P_j, for this hop, such that the honest parties will eventually hold $[c^{(j)}_{\mathsf{iter}}]$. Moreover, party P_j will be distinct from the summand-sharing party selected for any hop number $\mathsf{hop}' < \mathsf{hop}$.

Proof. The proof is identical to that of Claim 3.11, except that we now use Claim 4.15 to argue that for every ordered pair $(p, q) \in \mathsf{SIS}_{\mathsf{iter}}$, there exists at least one *honest* party in $(S_p \cap S_q) \setminus \mathcal{GD}$, say P_k, who will have both the shares $[a]_p$ as well as $[b_q]$ (and hence the summand $[a]_p[b]_q$). □

Claim 4.17. In protocol $\Pi_{\mathsf{BasicMult}}$, all honest parties eventually obtain an output. The protocol makes $\mathcal{O}(n^2)$ calls to $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$.

Proof. The proof is similar to that of Claim 3.12. □

Claim 4.18. During protocol $\Pi_{\mathsf{BasicMult}}$, Adv learns nothing about a and b.

Proof. The proof is similar to that of Claim 3.14. □

Claim 4.19. In $\Pi_{\mathsf{BasicMult}}$, if no party in $\mathcal{P} \setminus \mathcal{GD}$ behaves maliciously, then for each $P_i \in \mathsf{Selected}_{\mathsf{iter}}$, the condition $c^{(i)} = \sum_{(p,q) \in \mathsf{SIS}^{(i)}_{\mathsf{iter}}} [a]_p[b]_q$ holds, which further implies that $c = ab$ holds.

Proof. The proof is similar to that of Claim 3.13. □

Lemma 4.20 now follows from Claims 4.15-4.19.

Lemma 4.20. *Let $\mathcal{Z}$ satisfy the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition and let $\mathbb{S} = (S_1, \ldots, S_h) = \{\mathcal{P} \setminus Z | Z \in \mathcal{Z}\}$. Consider an arbitrary* iter*, such that all honest parties participate in the instance $\Pi_{\mathsf{BasicMult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], \mathcal{GD}, \mathsf{iter})$. Then all honest parties eventually compute $[c_{\mathsf{iter}}]$ and $[c^{(1)}_{\mathsf{iter}}], \ldots, [c^{(n)}_{\mathsf{iter}}]$ where $c_{\mathsf{iter}} = c^{(1)}_{\mathsf{iter}} + \ldots + c^{(n)}_{\mathsf{iter}}$, provided no honest party is ever included in the $\mathcal{GD}$. If no party in $\mathcal{P} \setminus \mathcal{GD}$ behaves maliciously, then $c_{\mathsf{iter}} = ab$ holds. In the protocol,* Adv *does not learn any additional information about a and b. The protocol makes $\mathcal{O}(n^2)$ calls to $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$.*

We claim another property of $\Pi_{\mathsf{BasicMult}}$, which will be useful later while analyzing the properties of $\Pi_{\mathsf{RandMultCI}}$, where $\Pi_{\mathsf{BasicMult}}$ is used as a sub-protocol.

Claim 4.21. For any iter, if $P_j \in \mathsf{Selected}_{\mathsf{iter}}$ during the instance $\Pi_{\mathsf{BasicMult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a], [b], \mathcal{GD}, \mathsf{iter})$, then $P_j \notin \mathcal{GD}$.

Proof. The proof is similar to that of Claim 3.16. □

We finally end this section by discussing the modifications to the protocol $\Pi_{\mathsf{BasicMult}}$ for handling M pairs of inputs.

Protocol $\Pi_{\mathsf{BasicMult}}$ for M pairs of inputs. Protocol $\Pi_{\mathsf{BasicMult}}$ can be easily modified if executed with input $\{([a^{(\ell)}], [b^{(\ell)}])\}_{\ell=1,\ldots,M}$. The modifications will be along similar lines to those done for Π_{OptMult}. Consequently, there will be $\mathcal{O}(n^2 M)$ calls to $\mathcal{F}_{\mathsf{VSS}}$, but *only* $\mathcal{O}(n^2)$ calls to $\mathcal{F}_{\mathsf{ABA}}$.

4.3.2 Protocol $\Pi_{\mathsf{RandMultCI}}$ for Detectable Random-Triple Generation

Based on $\Pi_{\mathsf{BasicMult}}$, we design a protocol $\Pi_{\mathsf{RandMultCI}}$, which takes as input an iteration number iter and an existing set of *corrupt* parties $\mathcal{GD}$. If no party in $\mathcal{P} \setminus \mathcal{GD}$ behaves maliciously, then the protocol outputs a random secret-shared multiplication-triple $[a_{\mathsf{iter}}], [b_{\mathsf{iter}}], [c_{\mathsf{iter}}]$. Else, except with probability $\frac{1}{|\mathbb{F}|}$, the parties update $\mathcal{GD}$ by identifying at least one *new* corrupt party among $\mathcal{P} \setminus \mathcal{GD}$. The protocol is based on a *synchronous* protocol from [24] with similar properties.

In the protocol, the parties first generate secret-sharing of random values $a_{\mathsf{iter}}, b_{\mathsf{iter}}, b'_{\mathsf{iter}}$ and r_{iter}. While a_{iter} and b_{iter} are the candidate pair of values for the multiplication triple returned by $\Pi_{\mathsf{RandMultCI}}$, the values b'_{iter} and r_{iter} are used to identify if any cheating has occurred. Two instances of $\Pi_{\mathsf{BasicMult}}$ with inputs $[a_{\mathsf{iter}}], [b_{\mathsf{iter}}]$ and $[a_{\mathsf{iter}}], [b'_{\mathsf{iter}}]$ are run to obtain $[c_{\mathsf{iter}}]$ and $[c'_{\mathsf{iter}}]$ respectively. The parties then reconstruct the "challenge" r_{iter} and *publicly* check if $[a_{\mathsf{iter}}](r_{\mathsf{iter}}[b_{\mathsf{iter}}] + [b'_{\mathsf{iter}}]) = (r_{\mathsf{iter}}[c_{\mathsf{iter}}] + [c'_{\mathsf{iter}}])$ holds, which should be the case if *no* cheating has occurred during the instances of $\Pi_{\mathsf{BasicMult}}$. If the condition holds, then the parties output $[a_{\mathsf{iter}}], [b_{\mathsf{iter}}], [c_{\mathsf{iter}}]$, which is guaranteed to be a multiplication-triple, except with probability $\frac{1}{|\mathbb{F}|}$. Otherwise, the parties proceed to identify at least one new corrupt party by reconstructing $[a_{\mathsf{iter}}], [b_{\mathsf{iter}}], [b'_{\mathsf{iter}}], [c_{\mathsf{iter}}], [c'_{\mathsf{iter}}]$ and the sum of the summands shared by the various summand-sharing parties during the instances of $\Pi_{\mathsf{BasicMult}}$.

Protocol $\Pi_{\mathsf{RandMultCI}}$ is formally presented in Fig 21.

Protocol $\Pi_{\mathsf{RandMultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, \mathcal{GD}, \mathsf{iter})$

– **Generating Secret-Sharing of Random Values**: The parties jointly generate $[a_{\mathsf{iter}}], [b_{\mathsf{iter}}], [b'_{\mathsf{iter}}]$ and $[r_{\mathsf{iter}}]$, where $a_{\mathsf{iter}}, b_{\mathsf{iter}}, b'_{\mathsf{iter}}$ and r_{iter} are random from the view-point of Adv, by using a similar procedure as in $\Pi_{\mathsf{PerTriples}}$. For this, each $P_i \in \mathcal{P}$ acts as a dealer, picks random $a^{(i)}_{\mathsf{iter}}, b^{(i)}_{\mathsf{iter}}, b'^{(i)}_{\mathsf{iter}}, r^{(i)}_{\mathsf{iter}}$ from $\mathbb{F}$ and generates random $[a^{(i)}_{\mathsf{iter}}], [b^{(i)}_{\mathsf{iter}}], [b'^{(i)}_{\mathsf{iter}}]$ and $[r^{(i)}_{\mathsf{iter}}]$, by making calls to $\mathcal{F}_{\mathsf{VSS}}$. The parties then agree on a common subset of parties $\mathcal{CS}$ through ACS as in $\Pi_{\mathsf{PerTriples}}$, such that $\mathcal{P} \setminus \mathcal{CS} \in \mathcal{Z}$ and for each $P_j \in \mathcal{CS}$, the honest parties eventually hold $[a^{(j)}_{\mathsf{iter}}], [b^{(j)}_{\mathsf{iter}}], [b'^{(j)}_{\mathsf{iter}}]$ and $[r^{(j)}_{\mathsf{iter}}]$. The parties then set

$$[a_{\mathsf{iter}}] \stackrel{def}{=} \sum_{P_j \in \mathcal{CS}} [a^{(j)}_{\mathsf{iter}}], \quad [b_{\mathsf{iter}}] \stackrel{def}{=} \sum_{P_j \in \mathcal{CS}} [b^{(j)}_{\mathsf{iter}}], \quad [b'_{\mathsf{iter}}] \stackrel{def}{=} \sum_{P_j \in \mathcal{CS}} [b'^{(j)}_{\mathsf{iter}}] \quad \text{and } [r_{\mathsf{iter}}] \stackrel{def}{=} \sum_{P_j \in \mathcal{CS}} [r^{(j)}_{\mathsf{iter}}].$$

– **Running Multiplication Protocol and Reconstructing the Random Challenge**:
  - The parties participate in instances $\Pi_{\mathsf{BasicMult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a_{\mathsf{iter}}], [b_{\mathsf{iter}}], \mathcal{GD}, \mathsf{iter})$ and $\Pi_{\mathsf{BasicMult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a_{\mathsf{iter}}], [b'_{\mathsf{iter}}], \mathcal{GD}, \mathsf{iter})$ to get outputs $\{[c^{(1)}_{\mathsf{iter}}], \ldots, [c^{(n)}_{\mathsf{iter}}], [c_{\mathsf{iter}}]\}$ and $\{[c'^{(1)}_{\mathsf{iter}}], \ldots, [c'^{(n)}_{\mathsf{iter}}], [c'_{\mathsf{iter}}]\}$ respectively. Let $\mathsf{Selected}_{\mathsf{iter},c}$ and $\mathsf{Selected}_{\mathsf{iter},c'}$ be the summand-sharing parties for the two instances respectively. Moreover, for $P_j \in \mathsf{Selected}_{\mathsf{iter},c}$, let $\mathsf{SIS}^{(j)}_{\mathsf{iter},c}$ be the set of ordered pairs of indices corresponding to the summands whose sum has been shared by P_j during the instance $\Pi_{\mathsf{BasicMult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a_{\mathsf{iter}}], [b_{\mathsf{iter}}], \mathcal{GD}, \mathsf{iter})$. Similarly, for $P_j \in \mathsf{Selected}_{\mathsf{iter},c'}$, let $\mathsf{SIS}^{(j)}_{\mathsf{iter},c'}$ be the set of ordered pairs of indices corresponding to the summands whose sum has been shared by P_j during the instance $\Pi_{\mathsf{BasicMult}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, [a_{\mathsf{iter}}], [b'_{\mathsf{iter}}], \mathcal{GD}, \mathsf{iter})$.
  - Once the parties obtain their respective outputs from the instances of $\Pi_{\mathsf{BasicMult}}$, they participate in an instance of Π_{PerRec} with shares corresponding to $[r_{\mathsf{iter}}]$, to reconstruct r_{iter}.

– **Detecting Errors in Instances of $\Pi_{\mathsf{BasicMult}}$:**
  - The parties locally compute $[e_{\mathsf{iter}}] \stackrel{def}{=} r_{\mathsf{iter}}[b_{\mathsf{iter}}] + [b'_{\mathsf{iter}}]$ and then participate in an instance of Π_{PerRec} with shares corresponding to $[e_{\mathsf{iter}}]$, to reconstruct e_{iter}.
  - The parties locally compute $[d_{\mathsf{iter}}] \stackrel{def}{=} e_{\mathsf{iter}}[a_{\mathsf{iter}}] - r_{\mathsf{iter}}[c_{\mathsf{iter}}] - [c'_{\mathsf{iter}}]$ and then participate in an instance of Π_{PerRec} with shares corresponding to $[d_{\mathsf{iter}}]$, to reconstruct d_{iter}.
  - **Output Computation in Case of Success:** If $d_{\mathsf{iter}} = 0$, then every party $P_i \in \mathcal{P}$ sets the Boolean variable $\mathsf{flag}^{(i)}_{\mathsf{iter}} = 0$ and outputs $\{([a_{\mathsf{iter}}]_q, [b_{\mathsf{iter}}]_q, [c_{\mathsf{iter}}]_q)\}_{P_i \in S_q}$.
  - **Cheater Identification in Case of Failure:** If $d_{\mathsf{iter}} \neq 0$, then every party $P_i \in \mathcal{P}$ sets the Boolean variable $\mathsf{flag}^{(i)}_{\mathsf{iter}} = 1$ and proceeds as follows.
    - Participate in instances of $\Pi_{\mathsf{PerRecShare}}$ to reconstruct the shares $\{[a_{\mathsf{iter}}]_q, [b_{\mathsf{iter}}]_q, [b'_{\mathsf{iter}}]_q\}_{S_q \in \mathbb{S}}$ and in instances of Π_{PerRec} to reconstruct $c^{(1)}_{\mathsf{iter}}, \ldots, c^{(n)}_{\mathsf{iter}}, c'^{(1)}_{\mathsf{iter}}, \ldots, c'^{(n)}_{\mathsf{iter}}$.
    - Set $\mathcal{GD} = \mathcal{GD} \cup \{P_i\}$, if $P_i \in \mathsf{Selected}_{\mathsf{iter},c} \cup \mathsf{Selected}_{\mathsf{iter},c'}$ and the following holds for P_i:

$$r_{\mathsf{iter}} \cdot c^{(i)}_{\mathsf{iter}} + c'^{(i)}_{\mathsf{iter}} \neq r_{\mathsf{iter}} \cdot \sum_{(p,q) \in \mathsf{SIS}^{(i)}_{\mathsf{iter},c}} [a_{\mathsf{iter}}]_p [b_{\mathsf{iter}}]_q + \sum_{(p,q) \in \mathsf{SIS}^{(i)}_{\mathsf{iter},c'}} [a_{\mathsf{iter}}]_p [b'_{\mathsf{iter}}]_q.$$

Figure 21: Detectable triple generation protocol in the $(\mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{ABA}})$-hybrid model

We now formally prove the properties of the protocol $\Pi_{\mathsf{RandMultCI}}$. While proving these properties, we will assume that $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. This further implies that $\mathbb{S} = (S_1, \ldots, S_h) \stackrel{def}{=} \{\mathcal{P} \setminus Z | Z \in \mathcal{Z}\}$ satisfies the $\mathbb{Q}^{(2)}(\mathbb{S}, \mathcal{Z})$ condition.

We first claim that the honest parties eventually compute $[a_{\mathsf{iter}}], [b_{\mathsf{iter}}], [b'_{\mathsf{iter}}]$ and $[r_{\mathsf{iter}}]$

Claim 4.22. Consider an arbitrary iter, such that all honest parties participate in the instance $\Pi_{\mathsf{RandMultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, \mathcal{GD}, \mathsf{iter})$, where no honest party is present in $\mathcal{GD}$. Then the honest parties eventually compute $[a_{\mathsf{iter}}], [b_{\mathsf{iter}}], [b'_{\mathsf{iter}}]$ and $[r_{\mathsf{iter}}]$.

Proof. The proof is similar to the proof of Claim 3.37. □

We next claim that all honest parties will eventually agree on whether the instances of $\Pi_{\mathsf{BasicMult}}$ in $\Pi_{\mathsf{RandMultCI}}$ has succeeded or failed.

Claim 4.23. Consider an arbitrary iter, such that all honest parties participate in the instance $\Pi_{\mathsf{RandMultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, \mathcal{GD}, \mathsf{iter})$, where no honest party is present in $\mathcal{GD}$. Then all honest parties eventually reconstruct a (common) value d_{iter}. Consequently, each honest P_i eventually sets $\mathsf{flag}^{(i)}_{\mathsf{iter}}$ to either 0 or 1.

Proof. From Claim 4.22, the honest parties eventually hold $[a_{\text{iter}}], [b_{\text{iter}}], [b'_{\text{iter}}]$ and $[r_{\text{iter}}]$. From Lemma 4.20, it follows that the honest parties eventually hold the outputs $\{[c^{(1)}_{\text{iter}}], \ldots, [c^{(n)}_{\text{iter}}], [c_{\text{iter}}]\}$ and $\{[c'^{(1)}_{\text{iter}}], \ldots, [c'^{(n)}_{\text{iter}}], [c'_{\text{iter}}]\}$ from the corresponding instances of $\Pi_{\mathsf{BasicMult}}$. From Lemma 3.9, the honest parties eventually reconstruct r_{iter} from the corresponding instance of Π_{PerRec}. From the linearity property of secret-sharing, it then follows that the honest parties eventually hold $[e_{\text{iter}}]$ and hence eventually reconstruct e_{iter} from the corresponding instance of Π_{PerRec}. Again, from the linearity property of secret-sharing, it follows that the honest parties eventually hold $[d_{\text{iter}}]$, followed by eventually reconstructing d_{iter} from the corresponding instance of Π_{PerRec}. Now based on whether d_{iter} is 0 or not, each *honest* P_i eventually sets $\mathsf{flag}^{(i)}_{\text{iter}}$ to either 0 or 1. $\square$

We next claim that if no party in $\mathcal{P} \setminus \mathcal{GD}$ behaves maliciously, then the honest parties eventually hold a secret-shared multiplication-triple.

Claim 4.24. Consider an arbitrary iter, such that all honest parties participate in the instance $\Pi_{\mathsf{RandMultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, \mathcal{GD}, \text{iter})$, where no honest party is present in $\mathcal{GD}$. If no party in $\mathcal{P} \setminus \mathcal{GD}$ behaves maliciously, then $d_{\text{iter}} = 0$ and the honest parties eventually hold $([a_{\text{iter}}], [b_{\text{iter}}], [c_{\text{iter}}])$, where $c_{\text{iter}} = a_{\text{iter}} \cdot b_{\text{iter}}$ holds.

Proof. If no party in $\mathcal{P} \setminus \mathcal{GD}$ behaves maliciously, then from Lemma 4.20, the honest parties eventually compute $[c_{\text{iter}}]$ and $[c'_{\text{iter}}]$ from the respective instances of $\Pi_{\mathsf{BasicMult}}$, such that $c_{\text{iter}} = a_{\text{iter}} \cdot b_{\text{iter}}$ and $c'_{\text{iter}} = a_{\text{iter}} \cdot b'_{\text{iter}}$ holds. From Claim 4.23, the honest parties will eventually reconstruct d_{iter}. Moreover, since $c_{\text{iter}} = a_{\text{iter}} \cdot b_{\text{iter}}$ and $c'_{\text{iter}} = a_{\text{iter}} \cdot b'_{\text{iter}}$ holds, the value d_{iter} will be 0 and consequently, the honest parties will output $([a_{\text{iter}}], [b_{\text{iter}}], [c_{\text{iter}}])$. $\square$

We next show that if $d_{\text{iter}} \neq 0$, then the honest parties eventually include at least one new maliciously-corrupt party in the set $\mathcal{GD}$.

Claim 4.25. Consider an arbitrary iter, such that all honest parties participate in the instance $\Pi_{\mathsf{RandMultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, \mathcal{GD}, \text{iter})$, where no honest party is present in $\mathcal{GD}$. If $d_{\text{iter}} \neq 0$, then the honest parties eventually update $\mathcal{GD}$ by adding a new maliciously-corrupt party in $\mathcal{GD}$.

Proof. Let $d_{\text{iter}} \neq 0$ and let $\mathsf{Selected}_{\text{iter}}$ be the set of summand-sharing parties across the two instances of $\Pi_{\mathsf{BasicMult}}$ executed in $\Pi_{\mathsf{RandMultCI}}$; i.e. $\mathsf{Selected}_{\text{iter}} \stackrel{def}{=} \mathsf{Selected}_{\text{iter},c} \cup \mathsf{Selected}_{\text{iter},c'}$. Note that there exists no $P_j \in \mathsf{Selected}_{\text{iter}}$ such that $P_j \in \mathcal{GD}$, which follows from Claim 4.21. We claim that there exists at least one party $P_j \in \mathsf{Selected}_{\text{iter}}$, such that corresponding to $c^{(j)}_{\text{iter}}$ and $c'^{(j)}_{\text{iter}}$, the following holds:

$$r_{\text{iter}} \cdot c^{(j)}_{\text{iter}} + c'^{(j)}_{\text{iter}} \neq r_{\text{iter}} \cdot \sum_{(p,q) \in \mathsf{SIS}^{(j)}_{\text{iter},c}} [a_{\text{iter}}]_p [b_{\text{iter}}]_q + \sum_{(p,q) \in \mathsf{SIS}^{(j)}_{\text{iter},c'}} [a_{\text{iter}}]_p [b'_{\text{iter}}]_q.$$

Assuming the above holds, the proof now follows from the fact that once the parties reconstruct $d_{\text{iter}} \neq 0$, they proceed to reconstruct the shares $\{[a_{\text{iter}}]_q, [b_{\text{iter}}]_q, [b'_{\text{iter}}]_q\}_{S_q \in \mathbb{S}}$ through appropriate instances of $\Pi_{\mathsf{PerRecShare}}$ and the values $c^{(1)}_{\text{iter}}, \ldots, c^{(n)}_{\text{iter}}, c'^{(1)}_{\text{iter}}, \ldots, c'^{(n)}_{\text{iter}}$ through appropriate instances of Π_{PerRec}. Upon reconstructing these values, party P_j will be eventually included in the set $\mathcal{GD}$. Moreover, it is easy to see that P_j is a maliciously-corrupt party, since for every *honest* $P_j \in \mathsf{Selected}_{\text{iter}}$, the following conditions hold:

$$c^{(j)}_{\text{iter}} = \sum_{(p,q) \in \mathsf{SIS}^{(j)}_{\text{iter},c}} [a_{\text{iter}}]_p [b_{\text{iter}}]_q \quad \text{and} \quad c'^{(j)}_{\text{iter}} = \sum_{(p,q) \in \mathsf{SIS}^{(j)}_{\text{iter},c'}} [a_{\text{iter}}]_p [b'_{\text{iter}}]_q.$$

We prove the above claim through a contradiction. So let the following condition hold for *each* $P_j \in \mathsf{Selected}_{\mathsf{iter}}$:

$$r_{\mathsf{iter}} \cdot c^{(j)}_{\mathsf{iter}} + c'^{(j)}_{\mathsf{iter}} = r_{\mathsf{iter}} \cdot \sum_{(p,q) \in \mathsf{SIS}^{(j)}_{\mathsf{iter},c}} [a_{\mathsf{iter}}]_p [b_{\mathsf{iter}}]_q + \sum_{(p,q) \in \mathsf{SIS}^{(j)}_{\mathsf{iter},c'}} [a_{\mathsf{iter}}]_p [b'_{\mathsf{iter}}]_q.$$

Next, summing the above equation over all $P_j \in \mathsf{Selected}_{\mathsf{iter}}$, we get that the following holds:

$$\sum_{P_j \in \mathsf{Selected}_{\mathsf{iter}}} r_{\mathsf{iter}} \cdot c^{(j)}_{\mathsf{iter}} + c'^{(j)}_{\mathsf{iter}} = \sum_{P_j \in \mathsf{Selected}_{\mathsf{iter}}} r_{\mathsf{iter}} \cdot \sum_{(p,q) \in \mathsf{SIS}^{(j)}_{\mathsf{iter},c}} [a_{\mathsf{iter}}]_p [b_{\mathsf{iter}}]_q + \sum_{(p,q) \in \mathsf{SIS}^{(j)}_{\mathsf{iter},c'}} [a_{\mathsf{iter}}]_p [b'_{\mathsf{iter}}]_q.$$

This implies that the following holds:

$$r_{\mathsf{iter}} \cdot \sum_{P_j \in \mathsf{Selected}_{\mathsf{iter}}} c^{(j)}_{\mathsf{iter}} + c'^{(j)}_{\mathsf{iter}} = r_{\mathsf{iter}} \cdot \sum_{P_j \in \mathsf{Selected}_{\mathsf{iter}}} \sum_{(p,q) \in \mathsf{SIS}^{(j)}_{\mathsf{iter},c}} [a_{\mathsf{iter}}]_p [b_{\mathsf{iter}}]_q + \sum_{(p,q) \in \mathsf{SIS}^{(j)}_{\mathsf{iter},c'}} [a_{\mathsf{iter}}]_p [b'_{\mathsf{iter}}]_q.$$

Now based on the way $a_{\mathsf{iter}}, b_{\mathsf{iter}}, b'_{\mathsf{iter}}, c_{\mathsf{iter}}$ and c'_{iter} are defined, the above implies that the following holds:

$$r_{\mathsf{iter}} \cdot c_{\mathsf{iter}} + c'_{\mathsf{iter}} = r \cdot a_{\mathsf{iter}} \cdot b_{\mathsf{iter}} + a_{\mathsf{iter}} \cdot b'_{\mathsf{iter}}$$

This further implies that

$$r_{\mathsf{iter}} \cdot c_{\mathsf{iter}} + c'_{\mathsf{iter}} = (r_{\mathsf{iter}} \cdot b_{\mathsf{iter}} + b'_{\mathsf{iter}}) \cdot a_{\mathsf{iter}}$$

Since in the protocol $e_{\mathsf{iter}} \stackrel{def}{=} r_{\mathsf{iter}} \cdot b_{\mathsf{iter}} + b'_{\mathsf{iter}}$, the above implies that

$$r_{\mathsf{iter}} \cdot c_{\mathsf{iter}} + c'_{\mathsf{iter}} = e_{\mathsf{iter}} \cdot a_{\mathsf{iter}} \quad \Rightarrow \; e_{\mathsf{iter}} \cdot a_{\mathsf{iter}} - r_{\mathsf{iter}} \cdot c_{\mathsf{iter}} - c'_{\mathsf{iter}} = 0 \quad \Rightarrow \; d_{\mathsf{iter}} = 0,$$

where the last equality follows from the fact that in the protocol, $d_{\mathsf{iter}} \stackrel{def}{=} e_{\mathsf{iter}} \cdot a_{\mathsf{iter}} - r_{\mathsf{iter}} \cdot c_{\mathsf{iter}} - c'_{\mathsf{iter}}$. However $d_{\mathsf{iter}} = 0$ is a contradiction, since according to the hypothesis of the claim, we are given that $d_{\mathsf{iter}} \neq 0$. □

We next show that if the honest parties output a secret-shared triple in the protocol, then except with probability $\frac{1}{|\mathbb{F}|}$, the triple is a multiplication-triple. Moreover, the triple will be random for the adversary.

Claim 4.26. Consider an arbitrary iter, such that all honest parties participate in the instance $\Pi_{\mathsf{RandMultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, \mathcal{GD}, \mathsf{iter})$, where no honest party is present in $\mathcal{GD}$. If $d_{\mathsf{iter}} = 0$, then the honest parties eventually output $([a_{\mathsf{iter}}], [b_{\mathsf{iter}}], [c_{\mathsf{iter}}])$, where except with probability $\frac{1}{|\mathbb{F}|}$, the condition $c_{\mathsf{iter}} = a_{\mathsf{iter}} \cdot b_{\mathsf{iter}}$ holds. Moreover, the view of Adv will be independent of $(a_{\mathsf{iter}}, b_{\mathsf{iter}}, c_{\mathsf{iter}})$.

Proof. Let $d_{\mathsf{iter}} = 0$. Then from the protocol steps, the honest parties eventually output $([a_{\mathsf{iter}}], [b_{\mathsf{iter}}], [c_{\mathsf{iter}}])$. In the protocol $d_{\mathsf{iter}} \stackrel{def}{=} e_{\mathsf{iter}} \cdot a_{\mathsf{iter}} - r_{\mathsf{iter}} \cdot c_{\mathsf{iter}} - c'_{\mathsf{iter}}$, where $e_{\mathsf{iter}} \stackrel{def}{=} r_{\mathsf{iter}} \cdot b_{\mathsf{iter}} + b'_{\mathsf{iter}}$. Since $d_{\mathsf{iter}} = 0$ holds, it implies that the honest parties have verified that the following holds:

$$r_{\mathsf{iter}}(a_{\mathsf{iter}} \cdot b_{\mathsf{iter}} - c_{\mathsf{iter}}) = (c'_{\mathsf{iter}} - a_{\mathsf{iter}} \cdot b'_{\mathsf{iter}}).$$

We also note that r_{iter} will be a random element from $\mathbb{F}$ and will be unknown to Adv till it is publicly reconstructed. This simply follows from the fact there will be at least one *honest* party P_j in the set $\mathcal{CS}$, such that the corresponding value $r^{(j)}_{\mathsf{iter}}$ shared by P_j will be random from the view-point of Adv. We also note that r_{iter} will be unknown to Adv, till the outputs for the underlying instances of $\Pi_{\mathsf{BasicMult}}$ are computed, and the honest parties hold $[c_{\mathsf{iter}}]$ and $[c'_{\mathsf{iter}}]$. This is because in the protocol, the honest parties start participating in the instance of Π_{PerRec} to reconstruct r_{iter}, only after they obtain their respective shares corresponding to $[c_{\mathsf{iter}}]$ and $[c'_{\mathsf{iter}}]$. Now we have the following cases with respect to whether any party from $\mathcal{P} \setminus \mathcal{GD}$ behaved maliciously during the underlying instances of $\Pi_{\mathsf{BasicMult}}$.

- **Case I:** $c_{\mathsf{iter}} = a_{\mathsf{iter}} \cdot b_{\mathsf{iter}}$ **and** $c'_{\mathsf{iter}} = a_{\mathsf{iter}} \cdot b'_{\mathsf{iter}}$ — In this case, $(a_{\mathsf{iter}}, b_{\mathsf{iter}}, c_{\mathsf{iter}})$ is a multiplication-triple.
- **Case II:** $c_{\mathsf{iter}} = a_{\mathsf{iter}} \cdot b_{\mathsf{iter}}$**, but** $c'_{\mathsf{iter}} \neq a_{\mathsf{iter}} \cdot b'_{\mathsf{iter}}$ — This case is never possible, as this will lead to the contradiction that $r_{\mathsf{iter}}(a_{\mathsf{iter}} \cdot b_{\mathsf{iter}} - c_{\mathsf{iter}}) \neq (c'_{\mathsf{iter}} - a_{\mathsf{iter}} \cdot b'_{\mathsf{iter}})$ holds.
- **Case III:** $c_{\mathsf{iter}} \neq a_{\mathsf{iter}} \cdot b_{\mathsf{iter}}$**, but** $c'_{\mathsf{iter}} = a_{\mathsf{iter}} \cdot b'_{\mathsf{iter}}$ — This case is possible only if $r_{\mathsf{iter}} = 0$, as otherwise this will lead to the contradiction that $r_{\mathsf{iter}}(a_{\mathsf{iter}} \cdot b_{\mathsf{iter}} - c_{\mathsf{iter}}) \neq (c'_{\mathsf{iter}} - a_{\mathsf{iter}} \cdot b'_{\mathsf{iter}})$ holds. However, since r_{iter} is a random element from $\mathbb{F}$, it implies that this case can occur only with probability at most $\frac{1}{|\mathbb{F}|}$.
- **Case IV:** $c_{\mathsf{iter}} \neq a_{\mathsf{iter}} \cdot b_{\mathsf{iter}}$ **as well as** $c'_{\mathsf{iter}} \neq a_{\mathsf{iter}} \cdot b'_{\mathsf{iter}}$ — This case is possible only if $r_{\mathsf{iter}} = (c'_{\mathsf{iter}} - a_{\mathsf{iter}} \cdot b'_{\mathsf{iter}}) \cdot (a_{\mathsf{iter}} \cdot b_{\mathsf{iter}} - c_{\mathsf{iter}})^{-1}$, as otherwise this will lead to the contradiction that $r_{\mathsf{iter}}(a_{\mathsf{iter}} \cdot b_{\mathsf{iter}} - c_{\mathsf{iter}}) \neq (c'_{\mathsf{iter}} - a_{\mathsf{iter}} \cdot b'_{\mathsf{iter}})$ holds. However, since r_{iter} is a random element from $\mathbb{F}$, it implies that this case can occur only with probability at most $\frac{1}{|\mathbb{F}|}$.

Hence, we have shown that except with probability at most $\frac{1}{|\mathbb{F}|}$, the triple $(a_{\mathsf{iter}}, b_{\mathsf{iter}}, c_{\mathsf{iter}})$ is a multiplication-triple. To complete the proof, we need to argue that the view of Adv in the protocol, will be independent of the triple $(a_{\mathsf{iter}}, b_{\mathsf{iter}}, c_{\mathsf{iter}})$. For this, we first note that $a_{\mathsf{iter}}, b_{\mathsf{iter}}$ and b'_{iter} will be random for the adversary. The proof for this is similar to that of Claim 3.39 and follows from the fact that there will be at least one *honest* party P_j in $\mathcal{CS}$, such that the corresponding values $a^{(j)}_{\mathsf{iter}}, b^{(j)}_{\mathsf{iter}}$ and $b'^{(j)}_{\mathsf{iter}}$ shared by P_j will be randomly distributed for Adv. From Lemma 4.20, Adv learns nothing additional about a_{iter}, b_{iter} and b'_{iter} during the two instances of $\Pi_{\mathsf{BasicMult}}$. While Adv learns the value of e_{iter}, since b'_{iter} is a uniformly distributed for Adv, for every candidate value of b'_{iter} from the view-point of Adv, there is a corresponding value of b_{iter} consistent with the e_{iter} learnt by Adv. Hence, learning e_{iter} does not add any new information about $(a_{\mathsf{iter}}, b_{\mathsf{iter}}, c_{\mathsf{iter}})$ to the view of Adv. Moreover, Adv will be knowing beforehand that d_{iter} will be 0 and hence, learning this value does not change the view of Adv regarding $(a_{\mathsf{iter}}, b_{\mathsf{iter}}, c_{\mathsf{iter}})$. □

We next derive the communication complexity of the protocol $\Pi_{\mathsf{RandMultCI}}$.

Claim 4.27. Protocol $\Pi_{\mathsf{RandMultCI}}$ requires $\mathcal{O}(n^2)$ calls to $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$, and additionally incurs a communication of $\mathcal{O}(|\mathcal{Z}| \cdot n^3 \log |\mathbb{F}|)$ bits.

Proof. Follows from the communication complexity of the protocol $\Pi_{\mathsf{BasicMult}}$ (Claim 4.17) and the fact that if $d_{\mathsf{iter}} \neq 0$, then the parties proceed to publicly reconstruct $\mathcal{O}(n)$ values through instances of Π_{PerRec} and publicly reconstruct $\mathcal{O}(|\mathbb{S}|)$ number of shares through instances of $\Pi_{\mathsf{PerRecShare}}$, where $|\mathbb{S}| = |\mathcal{Z}|$ for our sharing specification $\mathbb{S}$. □

The proof of Lemma 4.28 now follows from Claims 4.22-4.27.

Lemma 4.28. *Let $\mathcal{Z}$ satisfy the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition and let $\mathbb{S} = (S_1, \ldots, S_h) = \{\mathcal{P} \setminus Z | Z \in \mathcal{Z}\}$. Consider an arbitrary* iter*, such that all honest parties participate in the instance $\Pi_{\mathsf{RandMultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, \mathcal{GD}, \mathsf{iter})$, where no honest party is present in $\mathcal{GD}$. Then each honest P_i eventually sets $\mathsf{flag}^{(i)}_{\mathsf{iter}}$ to either 0 or 1. In the former case, the honest parties output $([a_{\mathsf{iter}}], [b_{\mathsf{iter}}], [c_{\mathsf{iter}}])$, such that with probability at least $1 - \frac{1}{|\mathbb{F}|}$, the condition $c_{\mathsf{iter}} = a_{\mathsf{iter}} \cdot b_{\mathsf{iter}}$ holds. Moreover, the view of* Adv *will be independent of the triple $(a_{\mathsf{iter}}, b_{\mathsf{iter}}, c_{\mathsf{iter}})$. In the latter case, the honest parties will eventually include at least one new maliciously-corrupt party P_j to $\mathcal{GD}$. The protocol makes $\mathcal{O}(n^2)$ calls to $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$, and additionally incurs a communication of $\mathcal{O}(|\mathcal{Z}| \cdot n^3 \log |\mathbb{F}|)$ bits.*

Protocol $\Pi_{\mathsf{RandMultCI}}$ for M Triples. The extension of $\Pi_{\mathsf{RandMultCI}}$ for generating M triples is straightforward. The parties first generate M random shared tuples $\{([a^{(\ell)}_{\mathsf{iter}}], [b^{(\ell)}_{\mathsf{iter}}], [b'^{(\ell)}_{\mathsf{iter}}])\}_{\ell=1,\ldots,M}$ and a *single* random challenge $[r_{\mathsf{iter}}]$. The parties then run $2M$ instances of $\Pi_{\mathsf{BasicMult}}$ to compute $\{([c^{(\ell)}_{\mathsf{iter}}], [c'^{(\ell)}_{\mathsf{iter}}])\}_{\ell=1,\ldots,M}$, followed by probabilistically checking if all the instances of $\Pi_{\mathsf{BasicMult}}$ are executed correctly, by using the *same* r_{iter} for all the instances. If cheating is detected in any of the instances, then the parties proceed further to identify at least one new maliciously-corrupt party and update $\mathcal{GD}$, as done in $\Pi_{\mathsf{RandMultCI}}$. The protocol makes $\mathcal{O}(n^2 \cdot M)$ calls to $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{O}(n^2)$ calls to $\mathcal{F}_{\mathsf{ABA}}$, and additionally incurs a communication of $\mathcal{O}((M \cdot |\mathcal{Z}| \cdot n^2 + |\mathcal{Z}| \cdot n^3) \log |\mathbb{F}|)$ bits.

4.4 Statistically-Secure Pre-Processing Phase Protocol

The statistically-secure pre-processing phase protocol $\Pi_{\mathsf{StatTriples}}$ proceeds in iterations, where in each iteration an instance of $\Pi_{\mathsf{RandMultCI}}$ is invoked, which either succeeds or fails. In case of success, the parties output the returned secret-shared multiplication-triples. Else, they continue to the next iteration. As a new corrupt party is discarded in each failed iteration, the protocol eventually outputs shared multiplication-triples. Protocol $\Pi_{\mathsf{StatTriples}}$ for generating a *single* shared multiplication-triple is formally presented in Fig 22. The only modification to generate M secret-shared triples will be to call $\Pi_{\mathsf{RandMultCI}}$ for generating M random triples.

Protocol $\Pi_{\mathsf{StatTriples}}(\mathcal{P}, \mathcal{Z}, \mathbb{S})$

- **Initialization**: Parties initialize $\mathcal{GD} = \emptyset$ and $\mathsf{iter} = 1$.
- **Detectable Triple Generation**: Parties participate in an instance $\Pi_{\mathsf{RandMultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, \mathcal{GD}, \mathsf{iter})$ with session id $\mathsf{sid}_{\mathsf{iter}} \overset{def}{=} \mathsf{sid}||\mathsf{iter}$. Each $P_i \in \mathcal{P}$ then proceeds as follows.
  - **Positive Output**: If $\mathsf{flag}^{(i)}_{\mathsf{iter}}$ is set to 0 during the instance $\Pi_{\mathsf{RandMultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, \mathcal{GD}, \mathsf{iter})$, then output the shares $\{([a_{\mathsf{iter}}]_q, [b_{\mathsf{iter}}]_q, [c_{\mathsf{iter}}]_q)\}_{P_i \in S_q}$ obtained during the instance of $\Pi_{\mathsf{RandMultCI}}$.
  - **Negative Output**: If $\mathsf{flag}^{(i)}_{\mathsf{iter}}$ is set to 1 during the instance $\Pi_{\mathsf{RandMultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, \mathcal{GD}, \mathsf{iter})$, then set $\mathsf{iter} = \mathsf{iter} + 1$ and go to the step **Detectable Triple Generation**.

Figure 22: A statistically-secure protocol for $\mathcal{F}_{\mathsf{Triples}}$ with $M = 1$ in $(\mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{ABA}})$-hybrid for session id sid

We next prove the security of the protocol $\Pi_{\mathsf{StatTriples}}$ in the $(\mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{ABA}})$-hybrid model.

Theorem 4.29. *Let $\mathcal{Z}$ satisfy the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. Then $\Pi_{\mathsf{StatTriples}}$ UC-securely realizes $\mathcal{F}_{\mathsf{Triples}}$ in the $(\mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{ABA}})$-hybrid model, except with error probability of at most $\frac{n}{|\mathbb{F}|}$. The protocol makes $\mathcal{O}(n^3 \cdot M)$ calls to $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{O}(n^3)$ calls to $\mathcal{F}_{\mathsf{ABA}}$, and additionally communicates $\mathcal{O}((M \cdot |\mathcal{Z}| \cdot n^3 + |\mathcal{Z}| \cdot n^4) \log |\mathbb{F}|)$ bits.*

By replacing the calls to $\mathcal{F}_{\mathsf{VSS}}$ with protocol Π_{SVSS} (along with the modifications discussed in Section 4.2.1), protocol $\Pi_{\mathsf{StatTriples}}$ UC-securely realizes $\mathcal{F}_{\mathsf{Triples}}$ in the $\mathcal{F}_{\mathsf{ABA}}$-hybrid model, except with error probability $n^3 \cdot \epsilon_{\mathsf{AICP}}$. The protocol makes $\mathcal{O}(n^3)$ calls to $\mathcal{F}_{\mathsf{ABA}}$ and additionally incurs a communication of $\mathcal{O}(M \cdot |\mathcal{Z}| \cdot n^9 \log |\mathbb{F}|)$ bits.

Proof. The communication complexity and the number of calls to $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$ simply follow from the communication complexity of $\Pi_{\mathsf{RandMultCI}}$ and the fact that there might be $\mathcal{O}(n)$ instances of $\Pi_{\mathsf{RandMultCI}}$ in the protocol. This is because from Lemma 4.28, if any instance of $\Pi_{\mathsf{RandMultCI}}$ fails, then at least one new *corrupt* party is globally discarded and included in $\mathcal{GD}$. Once all the corrupt parties are included in $\mathcal{GD}$, then from Claim 4.24, the next instance of $\Pi_{\mathsf{RandMultCI}}$ is bound to give the correct output.

We next prove the security. For the ease of explanation, we consider the case where only one multiplication-triple is generated in $\Pi_{\mathsf{StatTriples}}$; i.e. $M = 1$. The proof can easily be modified for any general M.

Let Adv be an arbitrary adversary, attacking the protocol $\Pi_{\mathsf{StatTriples}}$ by corrupting a set of parties $Z^\star \in \mathcal{Z}$, and let Env be an arbitrary environment. We show the existence of a simulator $\mathcal{S}_{\mathsf{StatTriples}}$ (Fig 23), such that for any $Z^\star \in \mathcal{Z}$, the outputs of the honest parties and the view of the adversary in the protocol $\Pi_{\mathsf{StatTriples}}$ is indistinguishable from the outputs of the honest parties and the view of the adversary in an execution in the ideal world involving $\mathcal{S}_{\mathsf{StatTriples}}$ and $\mathcal{F}_{\mathsf{Triples}}$, except with probability at most $\frac{n}{|\mathbb{F}|}$.

The high level idea of the simulator is very similar to that of the simulator for the protocol $\Pi_{\mathsf{PerTriples}}$ (see the proof of Theorem 3.36). Throughout the simulation, the simulator itself performs the role of the ideal functionalities $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$ whenever required and performs the role of the honest parties, exactly as per the steps of the protocol. In each iteration, the simulator simulates the actions of honest parties during the underlying instance of $\Pi_{\mathsf{RandMultCI}}$ by playing the role of the honest parties with random inputs. Once the simulator finds any iteration of $\Pi_{\mathsf{RandMultCI}}$ to be successful, the simulator learns the secret-sharing of the output triple of that iteration and sends the shares of this triple, corresponding to the corrupt parties to $\mathcal{F}_{\mathsf{Triples}}$, on the behalf of Adv.

Simulator $\mathcal{S}_{\mathsf{StatTriples}}$

$\mathcal{S}_{\mathsf{StatTriples}}$ constructs virtual real-world honest parties and invokes the real-world adversary Adv. The simulator simulates the view of Adv, namely its communication with Env, the messages sent by the honest parties, and the interaction with $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$. In order to simulate Env, the simulator $\mathcal{S}_{\mathsf{StatTriples}}$ forwards every message it receives from Env to Adv and vice-versa. The simulator then simulates the various stages of the protocol as follows.

- **Initialization**: On behalf of the honest parties, the simulator initializes $\mathcal{GD}$ to $\emptyset$ and iter to 1.
- **Detectable Triple Generation**: The simulator plays the role of the honest parties as per the protocol and interacts with Adv for an instance $\Pi_{\mathsf{RandMultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, \mathcal{GD}, \mathsf{iter})$. During this instance, the simulator simulates the interface for $\mathcal{F}_{\mathsf{ABA}}$ and $\mathcal{F}_{\mathsf{VSS}}$ for Adv during the underlying instances of $\Pi_{\mathsf{BasicMult}}$, by itself performing the role of $\mathcal{F}_{\mathsf{ABA}}$ and $\mathcal{F}_{\mathsf{VSS}}$. Next, based on whether the instance is successful or not, simulator does the following.
  - **If during the instance $\Pi_{\mathsf{RandMultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, \mathcal{GD}, \mathsf{iter})$, simulator has set $\mathsf{flag}^{(i)}_{\mathsf{iter}} = 0$, corresponding to any $P_i \notin Z^\star$**: In this case, let $([\widetilde{a}_{\mathsf{iter}}], [\widetilde{b}_{\mathsf{iter}}], [\widetilde{c}_{\mathsf{iter}}])$ be the output of the honest parties from the instance of $\Pi_{\mathsf{RandMultCI}}$. The simulator then sets $\{[\widetilde{a}_{\mathsf{iter}}]_q, [\widetilde{b}_{\mathsf{iter}}]_q, [\widetilde{c}_{\mathsf{iter}}]_q\}_{S_q \cap Z^\star \neq \emptyset}$ to be the shares corresponding to the parties in $Z^\star$ and goes to the step labelled **Interaction with $\mathcal{F}_{\mathsf{Triples}}$**.
  - **If during the instance $\Pi_{\mathsf{RandMultCI}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, \mathcal{GD}, \mathsf{iter})$, simulator has set $\mathsf{flag}^{(i)}_{\mathsf{iter}} = 1$, corresponding to any $P_i \notin Z^\star$**: In this case, the simulator sets $\mathsf{iter} = \mathsf{iter} + 1$ and goes to step labelled **Detectable Triple Generation**.
- **Interaction with $\mathcal{F}_{\mathsf{Triples}}$**: Let $\{[\widetilde{a}]_q, [\widetilde{b}]_q, [\widetilde{c}]_q\}_{S_q \cap Z^\star \neq \emptyset}$ be the shares set by the simulator corresponding to the parties in $Z^\star$. The simulator sends $(\mathsf{shares}, \mathsf{sid}, \{[\widetilde{a}]_q, [\widetilde{b}]_q, [\widetilde{c}]_q\}_{S_q \cap Z^\star \neq \emptyset})$ to $\mathcal{F}_{\mathsf{Triples}}$, on the behalf of Adv.

Figure 23: Simulator for the protocol $\Pi_{\mathsf{StatTriples}}$ where Adv corrupts the parties in set $Z^\star \in \mathcal{Z}$

We now prove a series of claims which will help us to finally prove the theorem. We first show that the view generated by $\mathcal{S}_{\mathsf{StatTriples}}$ for Adv is identically distributed to Adv's view during the real execution of $\Pi_{\mathsf{StatTriples}}$.

Claim 4.30. The view of Adv in the simulated execution with $\mathcal{S}_{\mathsf{PerTriples}}$ is identically distributed as the view of Adv in the real execution of $\Pi_{\mathsf{StatTriples}}$.

Proof. In both the real as well as simulated execution, the parties run an instance of $\Pi_{\mathsf{RandMultCI}}$ for each iteration iter, where in the simulated execution, the role of the honest parties is played by the simulator, including the role of $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$. Now, in either execution, if $\mathsf{flag}^{(i)}_{\mathsf{iter}}$ is set to 0 during some iteration iter corresponding to any *honest* P_i, then from Lemma 4.28, the view of Adv will be independent of the underlying triple and hence, will be identically distributed in both the executions. Else, in both executions, at least one new corrupt party gets discarded and the parties proceed to the next iteration. Hence, the view of Adv in both executions is identically distributed. □

We now show that conditioned on the view of Adv, the output of honest parties is identically distributed in the real execution of $\Pi_{\mathsf{StatTriples}}$ involving Adv, as well as in the ideal execution involving $\mathcal{S}_{\mathsf{StatTriples}}$ and $\mathcal{F}_{\mathsf{Triples}}$.

Claim 4.31. Conditioned on the view of Adv, the output of the honest parties is identically distributed in the real execution of $\Pi_{\mathsf{StatTriples}}$ involving Adv and in the ideal execution involving $\mathcal{S}_{\mathsf{StatTriples}}$ and $\mathcal{F}_{\mathsf{Triples}}$, except with probability at most $\frac{n}{|\mathbb{F}|}$.

Proof. Consider an arbitrary view View of Adv, generated as per some execution of $\Pi_{\mathsf{StatTriples}}$. From Lemma 4.28, in the real execution of $\Pi_{\mathsf{StatTriples}}$, during each iteration, all honest parties either obtain shares of a random multiplication triple, or discard a *new* maliciously-corrupt party. Since $|Z^\star| < n$, it will take less than n iterations to discard all the maliciously-corrupt parties. Furthermore, once all parties in $Z^\star$ are discarded, from Claim 4.24, the next instance of $\Pi_{\mathsf{RandMultCI}}$ will output a secret-shared multiplication-triple for the honest parties. Consequently, within n iterations, there will be some iteration iter, such that all honest parties P_i eventually set $\mathsf{flag}^{(i)}_{\mathsf{iter}}$ to 0 and output a secret-shared triple $([a_{\mathsf{iter}}], [b_{\mathsf{iter}}], [c_{\mathsf{iter}}])$. Moreover, from the union bound, it follows that except with probability at most $\frac{n}{|\mathbb{F}|}$, the triple $(a_{\mathsf{iter}}, b_{\mathsf{iter}}, c_{\mathsf{iter}})$ will be a multiplication-triple. Furthermore, from Lemma 4.28, the triple will be randomly distributed over $\mathbb{F}$.

To complete the proof, we show that conditioned on the shares $\{([a_{\mathsf{iter}}]_q, [b_{\mathsf{iter}}]_q, [c_{\mathsf{iter}}]_q)\}_{S_q \cap Z^\star \neq \emptyset}$ (which are determined by View), the honest parties output a secret-sharing of some random multiplication-triple in the simulated execution, which is consistent with the shares $\{([a_{\mathsf{iter}}]_q, [b_{\mathsf{iter}}]_q, [c_{\mathsf{iter}}]_q)\}_{S_q \cap Z^\star \neq \emptyset}$. However, this simply follows from the fact that in the simulated execution, $\mathcal{S}_{\mathsf{StatTriples}}$ sends the shares $\{([a_{\mathsf{iter}}]_q, [b_{\mathsf{iter}}]_q, [c_{\mathsf{iter}}]_q)\}_{S_q \cap Z^\star \neq \emptyset}$ to $\mathcal{F}_{\mathsf{Triples}}$ on the behalf of the parties in $Z^\star$, and as an output, $\mathcal{F}_{\mathsf{Triples}}$ generates a random secret-sharing of some random multiplication-triple consistent with these shares. □

The theorem now follows from Claim 4.30 and Claim 4.31. □

5 MPC Protocols in the Pre-Processing Model

The MPC protocol Π_{AMPC} in the pre-processing model is standard. The parties first generate secret-shared random multiplication-triples through $\mathcal{F}_{\mathsf{Triples}}$ in a *pre-processing* phase. Each party then randomly secret-shares its input for ckt through $\mathcal{F}_{\mathsf{VSS}}$ in an *input* phase. To avoid an indefinite wait, the parties agree on a common subset of parties $\mathcal{CS}$ where $\mathcal{P} \setminus \mathcal{CS} \in \mathcal{Z}$, whose inputs are eventually secret-shared, through ACS. The parties will eventually obtain some $\mathcal{CS}$, since the set of honest parties constitute a candidate $\mathcal{CS}$. Once $\mathcal{CS}$ is decided, for the remaining parties the parties take a default-sharing of 0 as the corresponding input. The next phase is the *circuit-evaluation* phase, where the parties jointly evaluate each gate in ckt in a secret-shared fashion by generating a secret-sharing of the gate-output from a secret-sharing of the gate-input(s). Linear gates are evaluated non-interactively due to the linearity of secret-sharing. To evaluate multiplication gates, the parties deploy Beaver's method [2], using the secret-shared multiplication-triples generated by $\mathcal{F}_{\mathsf{Triples}}$. Finally, the parties publicly reconstruct the secret-shared function output.

In the protocol, all honest parties may not reconstruct the function-output at the same "time" and different parties may be at different phases of the protocol, as the protocol is executed asynchronously. Consequently, a party, upon reconstructing the function-output, *cannot* afford to terminate immediately, as its presence and participation might be needed for the completion of various phases of the protocol by other honest parties. A standard trick to get around this problem in the AMPC protocols [21, 22, 14] is to have an additional *termination phase*, whose code is executed concurrently throughout the protocol to check if a party can "safely" terminate the protocol with the function output.

Protocol Π_{AMPC}

Pre-Processing Phase

1. Send $(\mathsf{triples}, \mathsf{sid}, P_i)$ to the functionality $\mathcal{F}_{\mathsf{Triples}}$.
2. Request output from $\mathcal{F}_{\mathsf{Triples}}$ until an output $(\mathsf{tripleshares}, \mathsf{sid}, \{[a^{(\ell)}]_q, [b^{(\ell)}]_q, [c^{(\ell)}]_q)\}_{\ell \in \{1,\ldots,M\}, P_i \in S_q})$ is received from $\mathcal{F}_{\mathsf{Triples}}$.

Input Phase

Once the output from $\mathcal{F}_{\mathsf{Triples}}$ is received, then proceed as follows.

- **Secret-sharing of the Inputs and Collecting Shares of Other Inputs**:
  1. Upon having the input $x^{(i)}$ for the function f, randomly select the shares $x_1^{(i)}, \ldots, x_h^{(i)} \in \mathbb{F}$, subject to the condition that $x^{(i)} = x_1^{(i)} + \ldots + x_h^{(i)}$. Send $(\mathsf{dealer}, \mathsf{sid}_i, P_i, (x_1^{(i)}, \ldots, x_h^{(i)}))$ to $\mathcal{F}_{\mathsf{VSS}}$, where $\mathsf{sid}_i \stackrel{def}{=} \mathsf{sid}||i$.[a]
  2. For $j = 1, \ldots, n$, request for output from $\mathcal{F}_{\mathsf{VSS}}$ with sid_j corresponding to P_j, until an output is received.
- **Selecting Common Input-Providers**:
  1. Upon receiving $(\mathsf{share}, \mathsf{sid}_j, P_j, \{[x^{(j)}]_q\}_{P_i \in S_q})$ from $\mathcal{F}_{\mathsf{VSS}}$ with sid_j, send $(\mathsf{vote}, \mathsf{sid}_j, 1)$ to $\mathcal{F}_{\mathsf{ABA}}$ with sid_j, where $\mathsf{sid}_j \stackrel{def}{=} \mathsf{sid}||j$.[b]
  2. For $j = 1, \ldots, n$, keep requesting for output from $\mathcal{F}_{\mathsf{ABA}}$ with sid_j until an output is received.
  3. Upon receiving $(\mathsf{decide}, \mathsf{sid}_j, 1)$ from $\mathcal{F}_{\mathsf{ABA}}$ with sid_j corresponding to each $P_j \in \mathcal{GP}_i$ such that $\mathcal{P} \setminus \mathcal{GP}_i \in \mathcal{Z}$, send $(\mathsf{vote}, \mathsf{sid}_j, 0)$ to every $\mathcal{F}_{\mathsf{ABA}}$ with sid_j for which no input has been provided yet.
  4. Once $(\mathsf{decide}, \mathsf{sid}_j, v_j)$ is received from $\mathcal{F}_{\mathsf{ABA}}$ with sid_j for every $j \in \{1, \ldots, n\}$, set $\mathcal{CS} = \{P_j : v_j = 1\}$.[c]
  5. Wait until $(\mathsf{share}, \mathsf{sid}_j, P_j, \{[x^{(j)}]_q\}_{P_i \in S_q})$ is received from $\mathcal{F}_{\mathsf{VSS}}$ for every $P_j \in \mathcal{CS}$. For every $P_j \notin \mathcal{CS}$, participate in an instance of Π_{PerDefSh} with public input 0 to generate a default secret-sharing of 0.

Circuit-Evaluation Phase

Evaluate each gate g in the circuit according to the topological ordering as follows, depending upon the type of g.

- **Addition Gate**: If g is an addition gate with inputs x, y and output z, then corresponding to every S_q such that $P_i \in S_q$, set $[z]_q = [x]_q + [y]_q$ as the share corresponding to z. Here $\{[x]_q\}_{P_i \in S_q}$ and $\{[y]_q\}_{P_i \in S_q}$ are P_i's shares corresponding to gate-inputs x and y respectively.
- **Multiplication Gate**: If g is the ℓ^{th} multiplication gate with inputs x, y and output z, where $\ell \in \{1, \ldots, M\}$, then do the following:
  1. Corresponding to every S_q such that $P_i \in S_q$, set $[d^{(\ell)}]_q \stackrel{def}{=} [x]_q - [a^{(\ell)}]_q$ and $[e^{(\ell)}]_q \stackrel{def}{=} [y]_q - [b^{(\ell)}]_q$, where $\{[x]_q\}_{P_i \in S_q}$ and $\{[y]_q\}_{P_i \in S_q}$ are P_i's shares corresponding to gate-inputs x and y respectively and $\{([a^{(\ell)}]_q, [b^{(\ell)}]_q, [c^{(\ell)}]_q)\}_{P_i \in S_q}$ are P_i's shares corresponding to the ℓ^{th} multiplication-triple.
  2. Participate in instances of Π_{PerRec} with shares $\{[d^{(\ell)}]_q\}_{P_i \in S_q}$ and $\{[e^{(\ell)}]_q\}_{P_i \in S_q}$ to publicly reconstruct $d^{(\ell)}$ and $e^{(\ell)}$, where $d^{(\ell)} \stackrel{def}{=} x - a^{(\ell)}$ and $e^{(\ell)} \stackrel{def}{=} y - b^{(\ell)}$.
  4. Upon reconstructing $d^{(\ell)}$ and $e^{(\ell)}$, corresponding to every S_q such that $P_i \in S_q$, set $[z]_q \stackrel{def}{=} d^{(\ell)} \cdot e^{(\ell)} + d^{(\ell)} \cdot [b^{(\ell)}]_q + e^{(\ell)} \cdot [a^{(\ell)}]_q + [c^{(\ell)}]_q$. Set $\{[z]_q\}_{P_i \in S_q}$ as the shares corresponding to z.
- **Output Gate**: If g is the output gate with output y, then participate in an instance of Π_{PerRec} with shares $\{[y]_q\}_{P_i \in S_q}$ to publicly reconstruct y.

Termination Phase

Concurrently execute the following steps during the protocol:

1. Upon computing the circuit-output y, send the message $(\mathsf{ready}, \mathsf{sid}, P_i, y)$ to every party in $\mathcal{P}$.
2. Upon receiving the message $(\mathsf{ready}, \mathsf{sid}, P_j, y)$ from a set of parties $\mathcal{A}$ such that $\mathcal{Z}$ satisfies $\mathbb{Q}^{(1)}(\mathcal{A}, \mathcal{Z})$ condition, send $(\mathsf{ready}, \mathsf{sid}, P_i, y)$ to every party in $\mathcal{P}$.
3. Upon receiving the message $(\mathsf{ready}, \mathsf{sid}, P_j, y)$ from a set of parties $\mathcal{W}$ such that $\mathcal{P} \setminus \mathcal{W} \in \mathcal{Z}$, output y and terminate.

[a] The notation sid_i is used here to distinguish among the n different calls to $\mathcal{F}_{\mathsf{VSS}}$.

[b] The notation sid_j is used here to distinguish among the n different calls to $\mathcal{F}_{\mathsf{ABA}}$.

[c] The parties need not have to explicitly check if $\mathcal{P} \setminus \mathcal{CS} \in \mathcal{Z}$, since the way $\mathcal{CS}$ is computed, it will be ensured that $\mathcal{P} \setminus \mathcal{CS} \in \mathcal{Z}$.

Figure 24: The perfectly-secure AMPC protocol in the $(\mathcal{F}_{\mathsf{Triples}}, \mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{ABA}})$-hybrid model. The public inputs of the protocol are $\mathcal{P}, \mathsf{ckt}, \mathcal{Z}$ and the sharing specification $\mathbb{S} = \{S_1, \ldots, S_h\} \stackrel{def}{=} \{\mathcal{P} \setminus Z | Z \in \mathcal{Z}\}$. The above steps are executed by every $P_i \in \mathcal{P}$

Intuitively, protocol Π_{AMPC} eventually terminates as the set $\mathcal{CS}$ is eventually decided. This is because even if the corrupt parties do not secret-share their inputs, the inputs of all honest parties are eventually secret-shared. Once $\mathcal{CS}$ is decided, the evaluation of each gate will be eventually completed: while the addition gates are evaluated non-interactively, the evaluation of multiplication gates requires reconstructing the corresponding masked gate-inputs which is eventually completed due to the reconstruction protocols. The privacy of the inputs of the honest parties in $\mathcal{CS}$ will be maintained as the sharing specification $\mathbb{S}$ is $\mathcal{Z}$-private. Moreover, the inputs of the corrupt parties in $\mathcal{CS}$ will be independent of the inputs of the honest parties in $\mathcal{CS}$, as inputs are secret-shared via calls to $\mathcal{F}_{\mathsf{VSS}}$. Finally, correctness holds since each gate is evaluated correctly. We next rigorously formalize this intuition by giving a formal security proof and show that the protocol Π_{AMPC} is perfectly-secure, if the parties have access to ideal functionalities $\mathcal{F}_{\mathsf{Triples}}, \mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$.

Theorem 5.1. *Protocol Π_{AMPC} UC-securely realizes the functionality $\mathcal{F}_{\mathsf{AMPC}}$ for securely computing f (see Fig 1) with perfect security in the $(\mathcal{F}_{\mathsf{Triples}}, \mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{ABA}})$-hybrid model, in the presence of a static malicious adversary characterized by an adversary-structure $\mathcal{Z}$ satisfying the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. The protocol makes one call to $\mathcal{F}_{\mathsf{Triples}}$ and $\mathcal{O}(n)$ calls to $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$ and additionally incurs a communication of $\mathcal{O}(M \cdot |\mathcal{Z}| \cdot n^2 \log |\mathbb{F}|)$ bits, where M is the number of multiplication gates in the circuit* ckt *representing f.*

Proof. The communication complexity in the $(\mathcal{F}_{\mathsf{Triples}}, \mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{ABA}})$-hybrid model follows from the fact that for evaluating each multiplication gate, the parties need to run 2 instances of the reconstruction protocol Π_{PerRec}.

For security, let Adv be an arbitrary real-world adversary corrupting the set of parties $Z^\star \in \mathcal{Z}$ and let Env be an arbitrary environment. We show the existence of a simulator $\mathcal{S}_{\mathsf{AMPC}}$, such that the output of honest parties and the view of the adversary in an execution of the real protocol with Adv is identical to the output in an execution with $\mathcal{S}_{\mathsf{AMPC}}$ involving $\mathcal{F}_{\mathsf{AMPC}}$ in the ideal model. This further implies that Env cannot distinguish between the two executions. The steps of the simulator are given in Fig 25.

The high level idea of the simulator is as follows. During the simulated execution, the simulator itself performs the role of the ideal functionalities $\mathcal{F}_{\mathsf{Triples}}, \mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{ABA}}$ whenever required. Performing the role of $\mathcal{F}_{\mathsf{Triples}}$ allows the simulator to learn the secret-sharing of all the multiplication-triples. During the input phase, whenever Adv secret-shares any value through $\mathcal{F}_{\mathsf{VSS}}$ on the behalf of a corrupt party, the simulator records this on the behalf of the corrupt party. This allows the simulator to learn the function-input of the corresponding corrupt party. On the other hand, for the *honest* parties, the simulator picks arbitrary values as their function-inputs and simulates the secret-sharing of those input values using random shares, as per $\mathcal{F}_{\mathsf{VSS}}$. To select the common input-providers during the simulated execution, the simulator itself performs the role of $\mathcal{F}_{\mathsf{ABA}}$ and simulates the honest parties as per the steps of the protocol and $\mathcal{F}_{\mathsf{ABA}}$. This allows the simulator to learn the common subset of input-providers $\mathcal{CS}$, which the simulator passes to the functionality $\mathcal{F}_{\mathsf{AMPC}}$. Notice that the function-inputs for each corrupt party in $\mathcal{CS}$ will be available with the simulator. This is because for every corrupt party P_j which is added to $\mathcal{CS}$, at least one honest party P_i should participate with input 1 in the corresponding call to $\mathcal{F}_{\mathsf{ABA}}$. This implies that the honest party P_i must have received the shares P_j sent to $\mathcal{F}_{\mathsf{VSS}}$ from $\mathcal{F}_{\mathsf{VSS}}$. Since in the simulation, the role of $\mathcal{F}_{\mathsf{VSS}}$ is played by the simulator, it implies that the full vector of shares provided by P_j to $\mathcal{F}_{\mathsf{VSS}}$ will be known to the simulator. Hence, along with $\mathcal{CS}$, the simulator can send the corresponding function-inputs of the corrupt parties in $\mathcal{CS}$ to $\mathcal{F}_{\mathsf{AMPC}}$. Upon receiving the function-output, the simulator simulates the steps of the honest parties for the gate evaluations as per the protocol. Finally, for the output gate, the simulator arbitrarily computes a secret-sharing of the function-output y received from $\mathcal{F}_{\mathsf{AMPC}}$, which is consistent with the shares which corrupt parties hold for the output-gate sharing. Then, on the behalf of the honest parties, the simulator sends the shares corresponding to the above sharing of y during the public reconstruction of y. This ensures that in the simulated execution, Adv learns the function-output y. For the termination phase, the simulator sends y on the behalf of honest parties.

Simulator $\mathcal{S}_{\mathsf{AMPC}}$

$\mathcal{S}_{\mathsf{AMPC}}$ constructs virtual real-world honest parties and invokes the real-world adversary Adv. The simulator simulates the view of Adv, namely its communication with Env, the messages sent by the honest parties, and the interaction with various functionalities. In order to simulate Env, the simulator $\mathcal{S}_{\mathsf{AMPC}}$ forwards every message it receives from Env to Adv and vice-versa. The simulator then simulates the various phases of the protocol as follows.

Pre-Processing Phase

Simulating the call to $\mathcal{F}_{\mathsf{Triples}}$: The simulator simulates the steps of $\mathcal{F}_{\mathsf{Triples}}$ by itself playing the role of $\mathcal{F}_{\mathsf{Triples}}$. Namely, it receives the shares corresponding to the parties in $Z^\star$ for each multiplication-triple from Adv and then randomly generates secret-sharing of M random multiplication-triples $\{(\widetilde{a}^{(\ell)}, \widetilde{b}^{(\ell)}, \widetilde{c}^{(\ell)})\}_{\ell=1,\dots,M}$ consistent with the provided shares. At the end of simulation of this phase, the simulator will know the entire vector of shares corresponding to the secret-sharing of all multiplication-triples.

Input Phase

- The simulator simulates the operations of the honest parties during the input phase, by randomly picking $\widetilde{x}^{(j)}$ as the input, for every $P_j \not\in Z^\star$, selecting random shares $\widetilde{x}_1^{(j)}, \dots, \widetilde{x}_h^{(j)}$ such that $\widetilde{x}^{(j)} = \widetilde{x}_1^{(j)} + \dots + \widetilde{x}_h^{(j)}$, and setting $[\widetilde{x}^{(j)}]_q = \widetilde{x}_q^{(j)}$, for $q = 1, \dots, h$. When Adv requests output from $\mathcal{F}_{\mathsf{VSS}}$ with sid_j on the behalf of any party $P_i \in Z^\star$, then the simulator responds with an output $(\mathsf{share}, \mathsf{sid}_j, P_j, \{[\widetilde{x}^{(j)}]_q\}_{P_i \in S_q})$ on the behalf of $\mathcal{F}_{\mathsf{VSS}}$.
- Whenever Adv sends $(\mathsf{dealer}, \mathsf{sid}_i, P_i, (x_1^{(i)}, \dots, x_h^{(i)}))$ to $\mathcal{F}_{\mathsf{VSS}}$ on the behalf of any $P_i \in Z^\star$, the simulator records the input $x^{(i)} \stackrel{def}{=} x_1^{(i)} + \dots + x_h^{(i)}$ on the behalf of P_i and sets $[x^{(i)}] = (x_1^{(i)}, \dots, x_h^{(i)})$.
- When the simulation reaches the "Selecting Common Input-Providers" stage, the simulator simulates the interface of $\mathcal{F}_{\mathsf{ABA}}$ to Adv by itself performing the role of $\mathcal{F}_{\mathsf{ABA}}$. When the first honest party completes the simulated input phase, $\mathcal{S}_{\mathsf{AMPC}}$ learns the set $\mathcal{CS}$.

Interaction with $\mathcal{F}_{\mathsf{AMPC}}$: Once the simulator learns $\mathcal{CS}$, it sends the input values $x^{(i)}$ that it has recorded on the behalf of each $P_i \in (Z^\star \cap \mathcal{CS})$, and the set of input-providers $\mathcal{CS}$ to $\mathcal{F}_{\mathsf{AMPC}}$. Upon receiving the output y from $\mathcal{F}_{\mathsf{AMPC}}$, the simulator starts the simulation of circuit-evaluation phase.

Circuit-Evaluation Phase

The simulator simulates the evaluation of each gate g in the circuit in topological order as follows:

- **Addition Gate**: Since this step involves local computation, the simulator does not have to simulate any messages on the behalf of the honest parties. The simulator locally adds secret-sharings corresponding to the gate-inputs and obtains a secret-sharing corresponding to the gate-output.
- **Multiplication Gate**: If g is the ℓ^{th} multiplication gate in the circuit, then the simulator takes the complete secret-sharing of the ℓ^{th} multiplication triple $(\widetilde{a}^{(\ell)}, \widetilde{b}^{(\ell)}, \widetilde{c}^{(\ell)})$ and computes the messages of the honest parties as per the steps of the protocol (by considering secret-sharing of the above multiplication-triple and secret-sharing of the gate-inputs), and sends them to Adv on the behalf of the honest parties as part of the instances of Π_{PerRec} protocol. Once the simulation of the circuit-evaluation phase is done, the simulator will know a secret-sharing corresponding to the gate-output.
- **Output Gate**: Let $[\widetilde{y}] = (\widetilde{y}_1, \dots, \widetilde{y}_h)$ be the secret-sharing corresponding to the output gate, available with $\mathcal{S}_{\mathsf{AMPC}}$ during the simulated circuit-evaluation. The simulator then randomly selects shares $\widetilde{\mathbf{y}}_1, \dots, \widetilde{\mathbf{y}}_h$ such that $\widetilde{\mathbf{y}}_1 + \dots + \widetilde{\mathbf{y}}_h = y$ and $\widetilde{\mathbf{y}}_q = \widetilde{y}_q$ corresponding to every $S_q \in \mathbb{S}$ where $S_q \cap Z^\star \neq \emptyset$. Then, as part of the instance of Π_{PerRec} protocol to reconstruct the function output, the simulator sends the shares $\{\widetilde{\mathbf{y}}_q\}_{S_q \in \mathbb{S}}$ to Adv on the behalf of the honest parties.

Termination Phase

The simulator sends a ready message for y to Adv on the behalf of $P_i \notin Z^\star$, if in the simulated execution, P_i has computed y.

Figure 25: Simulator for the protocol Π_{AMPC} where Adv corrupts the parties in set $Z^\star \in \mathcal{Z}$

We next prove a sequence of claims, which helps us to show that the joint distribution of the honest parties and the view of Adv is identical in both the real, as well as the ideal-world. We first claim that in any execution of Π_{AMPC}, a set $\mathcal{CS}$ is eventually generated. This automatically implies that the honest parties eventually possess a secret-sharing of M random multiplication-triples generated by $\mathcal{F}_{\mathsf{Triples}}$, as well as a secret-sharing of the inputs of the parties in $\mathcal{CS}$.

Claim 5.2. In any execution of Π_{AMPC}, a set $\mathcal{CS}$ is eventually generated where $\mathcal{P} \setminus \mathcal{CS} \in \mathcal{Z}$, such that for every $P_j \in \mathcal{CS}$, there exists some $x^{(j)}$ held by P_j which is eventually secret-shared.

Proof. As the proof of this claim is similar to the proof of Claim 3.37, we skip the formal proof. □

We next show that the view generated by $\mathcal{S}_{\mathsf{AMPC}}$ for Adv is identically distributed to Adv's view during the real execution of Π_{AMPC}.

Claim 5.3. The view of Adv in the simulated execution with $\mathcal{S}_{\mathsf{AMPC}}$ is identically distributed to the view of Adv in the real execution of Π_{AMPC}.

Proof. It is easy to see that the view of Adv during the pre-processing phase is identically distributed in both the executions. This is because in both the executions, Adv receives no messages from the honest parties and the steps of $\mathcal{F}_{\mathsf{Triples}}$ are executed by the simulator itself in the simulated execution. Namely, in both the executions, Adv's view consists of the shares of M random multiplication-triples corresponding to the parties in $Z^\star$. So, let us fix these shares. Conditioned on these shares, during the input phase, Adv learns the shares $\{[x^{(j)}]_q\}_{P_j \notin Z^\star, (S_q \cap Z^\star) \neq \emptyset}$ during the real execution corresponding to the parties $P_j \notin Z^\star$. In the simulated execution, it learns the shares $\{[\widetilde{x}^{(j)}]_q\}_{P_j \notin Z^\star, (S_q \cap Z^\star) \neq \emptyset}$. Since the sharing specification $\mathbb{S}$ is $\mathcal{Z}$-private and the vector of shares $(x_1^{(j)}, \ldots, x_h^{(j)})$ as well as $(\widetilde{x}_1^{(j)}, \ldots, \widetilde{x}_h^{(j)})$ are randomly chosen, it follows that the distribution of the shares $\{[x^{(j)}]_q\}_{P_j \notin Z^\star, (S_q \cap Z^\star) \neq \emptyset}$ as well as $\{[\widetilde{x}^{(j)}]_q\}_{P_j \notin Z^\star, (S_q \cap Z^\star) \neq \emptyset}$ is identical and independent of both $x^{(j)}$ as well as $\widetilde{x}^{(j)}$, so let us fix these shares. Since the role of $\mathcal{F}_{\mathsf{ABA}}$ is played by the simulator itself, it follows easily that the view of Adv during the selection of the set $\mathcal{CS}$ is identically distributed in both the real as well as the simulated execution.

During the evaluation of linear gates, no communication is involved. During the evaluation of multiplication gates, in the simulated execution, the simulator will know the secret-sharing associated with gate-inputs and also the secret-sharing of the associated multiplication-triple. Hence, the simulator correctly sends the shares corresponding to the values $d^{(\ell)}$ and $e^{(\ell)}$ as per the protocol on the behalf of the honest parties. Moreover, the values $d^{(\ell)}$ and $e^{(\ell)}$ will be randomly distributed for Adv in both the executions, since the underlying multiplication-triple is randomly distributed, conditioned on the shares of the corrupt parties. Thus, Adv's view during the evaluation of multiplications gates is identically distributed in both the executions.

For the output gate, the shares received by Adv in the real execution from the honest parties correspond to a secret-sharing of the function-output y. From the steps of $\mathcal{S}_{\mathsf{AMPC}}$, it is easy to see that the same holds even in the simulated execution, as $\mathcal{S}_{\mathsf{AMPC}}$ sends to Adv shares corresponding to a secret-sharing of y, which are consistent with the shares held by Adv. Hence, Adv's view is identically distributed in both the executions during the evaluation of output gate. Finally, it is easy to see that Adv's view is identically distributed in both the executions during the termination phase. This is because in both the executions, every honest party who has obtained the function output y, sends a ready message for y. □

We next claim that conditioned on the view of Adv (which is identically distributed in both the executions from the last claim), the output of the honest parties is identically distributed in both the worlds.

Claim 5.4. Conditioned on the view of Adv, the output of the honest parties is identically distributed in the real execution of Π_{AMPC} involving Adv, as well as in the ideal execution involving $\mathcal{S}_{\mathsf{AMPC}}$ and $\mathcal{F}_{\mathsf{AMPC}}$.

Proof. Let View be an arbitrary view of Adv, and let $\mathcal{CS}$ be the set of input-providers determined by View (from Claim 5.2, such a set $\mathcal{CS}$ is bound to exist). Moreover, according to View, for every $P_i \in \mathcal{CS}$, there exists some input $x^{(i)}$ such that the parties hold a secret-sharing of $x^{(i)}$. Furthermore, from Claim 5.3, if $P_i \in Z^\star$ then the corresponding secret-sharing is included in View. For $P_i \notin Z^\star$, the corresponding $x^{(i)}$ is uniformly distributed conditioned on the shares of $x^{(i)}$ available with Adv as determined by View. Let us fix the $x^{(i)}$ values corresponding to the parties in $\mathcal{CS}$ and denote the vector of values $x^{(i)}$, where $x^{(i)} = 0$ if $P_i \notin \mathcal{CS}$, by $\vec{x}$.

It is easy to see that in the ideal-world, the output of the honest parties is y, where $y \stackrel{def}{=} f(\vec{x})$. This is because $\mathcal{S}_{\mathsf{AMPC}}$ provides the identity of $\mathcal{CS}$ along with the inputs $x^{(i)}$ corresponding to $P_i \in (\mathcal{CS} \cap Z^\star)$ to $\mathcal{F}_{\mathsf{AMPC}}$. We now show that the honest parties eventually output y even in the real-world. For this, we argue that all the values during the circuit-evaluation phase of the protocol are correctly secret-shared. Since the evaluation of linear gates needs only local computation, it follows that the output of the linear gates will be correctly secret-shared. During the evaluation of a multiplication gate, the honest parties will hold a secret-sharing of the corresponding $d^{(\ell)}$ and $e^{(\ell)}$ values, as during the pre-processing phase, all the multiplication-triples are generated in a secret-shared fashion, since they are computed and distributed by $\mathcal{F}_{\mathsf{Triples}}$. Since $\mathbb{S}$ satisfies the $\mathbb{Q}^{(2)}(\mathbb{S}, \mathcal{Z})$ condition, the honest parties eventually get $d^{(\ell)}$ and $e^{(\ell)}$ through the instances of Π_{PerRec}. This automatically implies that the honest parties eventually hold a secret-sharing of y and reconstruct it correctly, as y is reconstructed through an instance of Π_{PerRec}. Hence, during the termination phase, every honest party will eventually send a ready message for y, while the parties in $Z^\star$ may send a ready message for $y' \neq y$. Since $Z^\star \in \mathcal{Z}$, it follows that no honest party ever sends a ready message for y'. Hence no honest party ever outputs y', as it will never receive the required number of ready messages for y'. Since the ready messages of the *honest* parties for y are eventually delivered to every honest party, it follows that eventually, all honest parties receive sufficiently many ready messages to obtain some output, even if the corrupt parties does not send the required messages.

Now let P_i be the *first honest* party to terminate the protocol with some output. From the above arguments, the output has to be y. This implies that P_i receives ready messages for y from a set of parties $\mathcal{P} \setminus Z$, for some $Z \in \mathcal{Z}$. Let $\mathcal{H}$ be the set of *honest* parties whose ready messages are received by P_i. It is easy to see that $\mathcal{H} \notin \mathcal{Z}$, as otherwise, $\mathcal{Z}$ does not satisfy the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. The ready messages of the parties in $\mathcal{H}$ are eventually delivered to every honest party and hence, *each* honest party (including P_i) eventually executes step 2 of the termination phase and sends a ready message for y. It follows that the ready messages of *all* honest parties $\mathcal{P} \setminus Z^\star$ are eventually delivered to every honest party (irrespective of whether Adv sends all the required messages), guaranteeing that all honest parties eventually obtain the output y. □

The theorem now follows from Claims 5.2-5.4. □

If $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition, then we can replace the calls to $\mathcal{F}_{\mathsf{Triples}}$ and $\mathcal{F}_{\mathsf{VSS}}$ with perfectly-secure protocol $\Pi_{\mathsf{PerTriples}}$ and Π_{PVSS} respectively and the calls to $\mathcal{F}_{\mathsf{ABA}}$ with the ABA protocol of [12]. The resultant protocol then achieves perfect security in the plain model. On the other hand, if $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition, then we can replace the calls to $\mathcal{F}_{\mathsf{Triples}}$ and $\mathcal{F}_{\mathsf{VSS}}$ with statistically-secure protocols $\Pi_{\mathsf{StatTriples}}$ and Π_{SVSS} respectively and the calls to $\mathcal{F}_{\mathsf{ABA}}$ with the ABA protocol of [11].[19] The resultant protocol then achieves statistical security in the plain model. To bound the error probability of the statistically-secure protocol by $2^{-\kappa}$, we select a finite field $\mathbb{F}$ such that $|\mathbb{F}| > n^4 2^{\kappa}$. Based on the above discussion, we get the following corollaries of Theorem 5.1.

Corollary 5.5. *If $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition, then Π_{AMPC} UC-securely realizes $\mathcal{F}_{\mathsf{AMPC}}$ in plain model. The protocol incurs a total communication of $\mathcal{O}(M \cdot (|\mathcal{Z}|^2 \cdot n^7 \log|\mathbb{F}| + |\mathcal{Z}| \cdot n^9 \log n) + |\mathcal{Z}|^2 \cdot (n^{11} \log|\mathbb{F}| + n^{13} \cdot (\log n + \log|\mathcal{Z}|)))$ bits, where M is the number of multiplication gates in* ckt.

Corollary 5.6. *If $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition, then Π_{AMPC} UC-securely realizes $\mathcal{F}_{\mathsf{AMPC}}$ in the $\mathcal{F}_{\mathsf{ABA}}$-hybrid model with statistical security. If $|\mathbb{F}| > n^5 2^{\kappa}$ for a given statistical-security parameter κ, then the error probability of the protocol is at most $2^{-\kappa}$. The protocol incurs a total communication of $\mathcal{O}(M \cdot |\mathcal{Z}| \cdot n^9 \log|\mathbb{F}| + |\mathcal{Z}| \cdot (n^{10} \log|\mathbb{F}| + n^{11} \log n))$ bits, where M is the number of multiplication gates in* ckt.

Acknowledgement. We would like to sincerely thank the anonymous referees for their insightful remarks and feedback, which helped us to significantly improve the overall paper.

References

[1] I. Abraham, D. Dolev, and G. Stern. Revisiting Asynchronous Fault Tolerant Computation with Optimal Resilience. In *PODC*, pages 139–148. ACM, 2020.

[2] D. Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In J. Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991.

[3] Z. Beerliová-Trubíniová and M. Hirt. Simple and Efficient Perfectly-Secure Asynchronous MPC. In *ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 376–392. Springer Verlag, 2007.

[4] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous Secure Computation. In *STOC*, pages 52–61. ACM, 1993.

[5] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *STOC*, pages 1–10. ACM, 1988.

[6] M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous Secure Computations with Optimal Resilience (Extended Abstract). In *PODC*, pages 183–192. ACM, 1994.

[7] R. Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute, Israel, 1995.

[8] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.

[9] R. Canetti. Universally Composable Security. *J. ACM*, 67(5):28:1–28:94, 2020.

[19] Recall that the ABA protocols of [12], as well as [11], are almost-surely terminating, where $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ and $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition respectively. Communication-complexity-wise, the ABA protocol of [12] is more efficient, compared to [11].

[10] R. Canetti and T. Rabin. Fast Asynchronous Byzantine Agreement with Optimal Resilience. In *STOC*, pages 42–51, 1993.

[11] A. Choudhury. Almost-Surely Terminating Asynchronous Byzantine Agreement Against General Adversaries with Optimal Resilience. In *ICDCN*, pages 167–176. ACM, 2023.

[12] A. Choudhury and N. Pappu. Perfectly-Secure Asynchronous MPC for General Adversaries (Extended Abstract). In *INDOCRYPT*, volume 12578 of *Lecture Notes in Computer Science*, pages 786–809. Springer, 2020.

[13] A. Choudhury and A. Patra. An Efficient Framework for Unconditionally Secure Multiparty Computation. *IEEE Trans. Information Theory*, 63(1):428–468, 2017.

[14] R. Cohen. Asynchronous Secure Multiparty Computation in Constant Time. In *PKC*, volume 9615 of *Lecture Notes in Computer Science*, pages 183–207. Springer, 2016.

[15] S. Coretti, J. A. Garay, M. Hirt, and V. Zikas. Constant-Round Asynchronous Multi-Party Computation Based on One-Way Functions. In *ASIACRYPT*, volume 10032 of *Lecture Notes in Computer Science*, pages 998–1021, 2016.

[16] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient Multiparty Computations Secure Against an Adaptive Adversary. In *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 311–326. Springer, 1999.

[17] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM*, 32(2):374–382, 1985.

[18] M Fitzi and U. M. Maurer. Efficient Byzantine Agreement Secure Against General Adversaries. In *DISC*, volume 1499 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 1998.

[19] O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[20] O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In A. V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987.

[21] M. Hirt, J. B. Nielsen, and B. Przydatek. Cryptographic Asynchronous Multi-party Computation with Optimal Resilience (Extended Abstract). In R. Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 322–340. Springer, 2005.

[22] M. Hirt, J. B. Nielsen, and B. Przydatek. Asynchronous Multi-Party Computation with Quadratic Communication. In *ICALP*, volume 5126 of *Lecture Notes in Computer Science*, pages 473–485. Springer, 2008.

[23] Martin Hirt and Ueli Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, 2000.

[24] Martin Hirt and Daniel Tschudi. Efficient general-adversary multi-party computation. In *ASIACRYPT*, volume 8270 of *Lecture Notes in Computer Science*, pages 181–200. Springer, 2013.

[25] M. Ito, A. Saito, and T. Nishizeki. Secret Sharing Schemes Realizing General Access Structures). In *Global Telecommunication Conference, Globecom*, pages 99–102. IEEE Computer Society, 1987.

[26] J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Universally Composable Synchronous Computation. In *TCC*, volume 7785 of *Lecture Notes in Computer Science*, pages 477–498. Springer, 2013.

[27] K. Kursawe and F. C. Freiling. Byzantine Fault Tolerance on General Hybrid Adversary Structures. Technical Report, RWTH Aachen, 2005.

[28] U. M. Maurer. Secure Multi-party Computation Made Simple. In *SCN*, volume 2576 of *Lecture Notes in Computer Science*, pages 14–28. Springer, 2002.

[29] A. Patra, A. Choudhury, and C. Pandu Rangan. Asynchronous Byzantine Agreement with Optimal Resilience. *Distributed Computing*, 27(2):111–146, 2014.

[30] A. Patra, A. Choudhury, and C. Pandu Rangan. Efficient Asynchronous Verifiable Secret Sharing and Multiparty Computation. *J. Cryptology*, 28(1):49–109, 2015.

[31] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching Agreement in the Presence of Faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[32] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstract). In *STOC*, pages 73–85. ACM, 1989.

[33] A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979.

[34] A. C. Yao. Protocols for Secure Computations (Extended Abstract). In *FOCS*, pages 160–164. IEEE Computer Society, 1982.