Best of Both Worlds
Revisiting the Spymasters Double Agent Problem

Anasuya Acharya[1], Carmit Hazay[2], Oxana Poburinnaya[3], and Muthuramakrishnan Venkitasubramaniam[4]

[1] Bar-Ilan University
acharya@biu.ac.il
[2] Bar-Ilan University
carmit.hazay@biu.ac.il
[3] oxanapob@bu.edu
[4] Georgetown University
mv783@georgetown.edu

Abstract. This work introduces the notion of secure multiparty computation: MPC with fall-back security. Fall-back security for an n-party protocol is defined with respect to an adversary structure $\mathcal{Z}$ wherein security is guaranteed in the presence of both a computationally unbounded adversary with adversary structure $\mathcal{Z}$, and a computationally bounded adversary corrupting an arbitrarily large subset of the parties. This notion was considered in the work of Chaum (Crypto 89) via the Spymaster's double agent problem where he showed a semi-honest secure protocol for the honest majority adversary structure.
Our first main result is a compiler that can transform any n-party protocol that is semi-honestly secure with statistical security tolerating an adversary structure $\mathcal{Z}$ to one that (additionally) provides semi-honest fall-back security w.r.t $\mathcal{Z}$. The resulting protocol has optimal round complexity, up to a constant factor, and is optimal in assumptions and the adversary structure. Our second result fully characterizes when malicious fall-back security is feasible. More precisely, we show that malicious fall-back secure protocol w.r.t $\mathcal{Z}$ exists if and only if $\mathcal{Z}$ admits unconditional MPC against a semi-honest adversary (namely, iff $\mathcal{Z} \in \mathcal{Q}^2$).

Keywords: MPC with Fall-back Security· Best of Both Worlds · MPC Protocols Compiler

1 Introduction

The problem of secure multiparty computation (MPC) considers a set of parties with private inputs that wish to jointly compute some function of their inputs while preserving certain security properties, like privacy (nothing but the output is learned), and correctness (output is computed correctly according to the specified function). These properties are required to hold in the presence of an adversary that controls a subset of the parties and launches an attack on the protocol in an attempt to breach its security (e.g., to learn more than it should about the honest parties' inputs).

A standard classification distinguishes adversaries that are *computationally unbounded* from those that are *computationally bounded* (i.e. probabilistic polynomial time algorithms). MPC protocols secure against the former can be designed only when the adversary corrupts fewer than half of the parties but *unconditionally*, whereas, MPC protocols secure against the latter can be designed for arbitrary corruptions (i.e. up to all-but-one of the parties) but require making *cryptographic assumptions* (such as the hardness of factoring or discrete logarithm). In this work, we revisit the question of achieving the "best of both worlds", as considered in the work of Chaum [Cha89] motivated by the following scenarios:

1. **Bringing people of different beliefs.** Is it possible to design an MPC protocol where some of the parties want unconditional security while some parties demand security against arbitrary collusion?
2. **Resistance to future attacks.** Can we design an MPC protocol with security against arbitrary collusion that offers some security even when the underlying cryptographic assumption is broken? For example, can we build an MPC based on *quantum un-safe* primitives that can offer some protection should quantum computers become feasible?
3. **David versus many Goliaths.** Can we design an n party MPC protocol where a single (designated) party can be unconditionally protected against an (unbounded) adversary colluding with the rest of the world (i.e. remaining $n-1$ parties)?

We answer all these questions in the affirmative by studying MPC protocols that offer *best of both worlds* security, namely, unconditional security against minority collusion and computational security against arbitrary collusion.

Chaum in [Cha89] motivated this notion of security via the "Spymasters double agent problem" where a set of countries are willing to perform an MPC to identify "double agents". The main concern for spymasters is that a majority of the countries could collude or a minority of them could break the cryptosystem to uncover secrets. In the same work, Chaum designed an "optimal" n-party MPC protocol in the passive (semi-honest) setting, namely, security holds unconditionally against fewer than $n/2$ corruptions whereas computational security holds against arbitrary corruptions in the presence of a passive adversary. In the active setting, Chaum constructed a protocol with unconditional security against fewer than $n/3$ corruptions and computational security against fewer than $n/2$ corruptions. However, this protocol was subsumed by later results that provided a stronger security guarantee, namely, unconditional security against fewer than $n/2$ corruptions. Thus, the main challenge of designing an MPC protocol with best-of-both-worlds security against an active adversary still remains open. More precisely, we investigate in this work the following question:

> *Under what circumstances can we achieve best of both worlds security against an active adversary?*

1.1 Our Contributions

In this work, we completely characterize when best of both worlds security is feasible. Roughly speaking, any adversary structure that admits an information-theoretically secure MPC against a passive adversary can be compiled into one that simultaneously provides information-theoretically security against an active adversary w.r.t the same adversary structure and additionally provides computational security against arbitrary corruptions. In fact, for this reason, we call our best-of-both-worlds security notion as *fall-back security.*

MPC WITH FALL-BACK SECURITY. We introduce a new notion of secure multiparty computation: MPC with fall-back security. Fall-back security for an n-party protocol is defined with respect to an adversary structure $\mathcal{Z}_{\mathsf{S}}$ wherein security is guaranteed in the presence of both a computationally unbounded adversary with adversary structure $\mathcal{Z}_{\mathsf{S}}$, and a computationally bounded adversary corrupting an arbitrary subset of the parties. We consider this definition in the context of semi-honest and malicious adversaries.

FALL-BACK SECURITY W.R.T SEMI-HONEST ADVERSARIES. We show that for any adversary structure $\mathcal{Z}_{\mathsf{S}}$ that admits an MPC protocol with unconditional security in the presence of a passive adversary, we can compile it to another MPC protocol that additionally provides computational security w.r.t arbitrary corruption of parties (again with a passive adversary). We build our compiler in two steps:

- First, we design an MPC protocol for a specific adversary structure. In fact, this will correspond to the "David vs many Goliaths" setting. More precisely, consider the adversary structure $\mathcal{Z}$ over n parties such that one designated party can never be corrupted. We describe an n-party protocol that can compute any functionality with semi-honest fall-back security tolerating $\mathcal{Z}$.
- Let Π_{stat} be an n-party protocol that is secure in the presence of a semi-honest unbounded adversary with adversary structure $\mathcal{Z}_{\mathsf{S}}$. We use the protocol designed in the first step as a building block to compile Π_{stat} into a protocol with semi-honest fall-back security tolerating $\mathcal{Z}_{\mathsf{S}}$.

More precisely, we achieve the following theorem:

Theorem (Informal) 1.1 *Assuming the existence of an r_{OT}-round oblivious transfer (OT) protocol with one-sided statistical security (i.e. statistical sender security or statistical receiver security) against a passive adversary. Then, any n-party function that can be securely implemented via an r-round n-party protocol Π_{stat}, that is secure in the presence of an unbounded semi-honest adversary w.r.t adversary structure $\mathcal{Z}_{\mathsf{S}}$, can be compiled to an $O(r \cdot r_{OT})$-round n-party protocol for the same functionality with semi-honest fall-back security tolerating $\mathcal{Z}_{\mathsf{S}}$.*

We remark that the existence of OT with one-sided statistical security is necessary and therefore minimal assumption. This result re-establishes the result of Chaum [Cha89] for threshold adversary structures. The main difference is that our construction is round optimal up to a constant factor, relies on the underlying assumption in a black-box manner, and generalizes to arbitrary adversary structures. Moreover, we achieve optimality in terms of adversary structures since we can compile any information-theoretic MPC for $\mathcal{Z}_{\mathsf{S}}$ to one with fall-back security. Recalling from [HM97] that an MPC protocol is feasible in the semi-honest setting iff the adversary structure $\mathcal{Z}_{\mathsf{S}} \in \mathcal{Q}^2$ where $\mathcal{Z}_{\mathsf{S}} \in \mathcal{Q}^2$ if for any two $Z_1, Z_2 \in \mathcal{Z}_{\mathsf{S}}$, it holds that $Z_1 \cup Z_2 \neq \{1, \ldots, n\}$. Thus, we have the corollary:

Corollary 1.2 *We can securely compute any functionality with semi-honest fall-back security w.r.t $\mathcal{Z}_{\mathsf{S}}$ if and only if $\mathcal{Z}_{\mathsf{S}} \in \mathcal{Q}^2$.*

Fall-Back Security w.r.t malicious adversaries. Next, we construct a protocol that can compute any functionality with maliciously secure fall-back security tolerating any possible adversary structure for statistical security. We use the SPDZ-type paradigm [DPSZ12], where we first generate authenticated Beaver triples and then use that to securely compute a function. Since the SPDZ online phase is essentially information-theoretically secure against arbitrary corruption by an unbounded adversary, it will suffice to design an MPC protocol with fall-back security with respect to malicious parties, for the authenticated Beaver triples functionality. We use a variant of the [HVW20] compiler (which, in turn, is a variant of [IPS08]) that "generically" compiles semi-honest protocols to malicious ones in a "modular" way. Formally, we prove the following theorem:

Theorem (Informal) 1.3 *Assume the existence of an OT protocol with one-sided statistical security against semi-honest adversaries. Then if the authenticated Beaver triples functionality can be securely implemented via a n-party protocol Π_{stat} secure in the presence of an unbounded semi-honest adversary w.r.t adversary structure $\mathcal{Z}_{\mathsf{S}}$, then any function can be compiled to an n-party protocol that computes the same functionality with malicious fall-back security tolerating $\mathcal{Z}_{\mathsf{S}}$.*

We remark that our result is optimal in terms of adversary structure and assumptions. Since any MPC protocol that is malicious fall-back secure w.r.t $\mathcal{Z}_{\mathsf{S}}$ is also information-theoretically secure w.r.t $\mathcal{Z}_{\mathsf{S}}$ against a passive adversary and as alluded before, implies OT with one-sided statistical security. Therefore, both assumptions are necessary. Thus, we have the following corollary.

Corollary 1.4 *We can securely compute any functionality with malicious fall-back security w.r.t $\mathcal{Z}_{\mathsf{S}}$ if and only if $\mathcal{Z}_{\mathsf{S}} \in \mathcal{Q}^2$.*

We remark here that [HM97] showed that malicious security in the information-theoretic setting is achievable iff $\mathcal{Z}_{\mathsf{S}} \in \mathcal{Q}^3$. Our result does not violate this since we only achieve security w.r.t abort in both the information-theoretic and computational settings.

1.2 Related Work

Several variants of best-of-both-worlds notions have been considered in the literature. The works of [Kat07, IKK^{+}11] consider achieving *full security* and is closest to our security notion. In these works, the authors consider an MPC protocol that provides guaranteed output delivery when there is an honest majority and security with abort against arbitrary corruption. They identify the necessary conditions for feasibility and design a protocol that offers only computational security in both corruption scenarios. Our work, in contrast, does not offer guaranteed output delivery but achieves unconditional security when a minority of the parties are corrupted.

In [KM20], Khurana and Mughees studied the round complexity of designing a 2PC protocol that offers unconditional security against one party and computational against the other (which is optimal as we cannot achieve statistical security against both parties). They showed that 4 rounds are sufficient to design such a protocol where the output is delivered to one designated party and 5 rounds if both parties receive output. This is optimal as previous works have shown matching lower bounds that hold even for the standard computational security against both parties [KO04,GMPP16]. This work has been improved more recently by [BPS22] showing a wide variety of assumptions to design such a protocol. Our work can be viewed as a generalization of their work from two-party to multiparty where even feasibility was previously unknown.

The work of Chaum et al. [CDvdG87] shows how to construct a protocol in the dishonest majority setting where one designated party's view is unconditionally hidden from the rest of the parties. Their protocol solves our third motivating question of David versus many Goliaths, however, their protocol is based on a specific assumption and proceeds in a round-robin fashion. Our protocol is optimal in round complexity up to a constant factor, and can be based on the minimal assumption of existence of oblivious transfer with one-sided statistical security.

1.3 Technical Overview

We define our notion of MPC with fall-back security. We also construct protocols that satisfy this in the presence of semi-honest and malicious adversaries.

FALL-BACK SECURITY. For an n-party functionality $\mathcal{F}$, we define the notion of Fall-Back security with respect to an adversary structure $\mathcal{Z}_{\mathsf{S}}$. Let $\mathcal{P}$ be the set of n parties executing a protocol to realize this functionality. We say that the protocol achieves fall-back security if it is secure in the presence of a computationally unbounded adversary corrupting any subset of parties $Z \in \mathcal{Z}_{\mathsf{S}}$. The protocol is additionally secure in the presence of a PPT adversary corrupting any $Z \subset \mathcal{P}$. We state formally two flavours of fall-back security, one in the presence of semi-honest adversaries and one in the presence of malicious adversaries in Section 3.

PROTOCOLS WITH FALL-BACK SECURITY: THE 2PC CASE. Before describing our n-party protocol that achieves fall-back security, let us examine the toy case of the 2-party setting. Recall that the standard 2-party Yao's Garbled circuit protocol where one party, referred to as the garbler, creates a garbled circuit computing the function and gives this to the other party, referred to as the evaluator, along with the input labels corresponding to its inputs. Then both parties engage in multiple instances of oblivious transfer (OT) through which the evaluator securely learns input labels corresponding to its inputs. The evaluator evaluates the garbled circuit and shares the output with the garbler.

In this protocol, suppose we instantiate the OT protocol with one that is secure in the presence of a semi-honest unbounded corrupt sender, then the view of the garbler contains the garbled circuit and input labels corresponding to its inputs (which are both independent of the evaluator's inputs) and the messages in the OT protocol that statistically secure against the sender. Therefore, we can argue that this protocol is secure against a semi-honest unbounded adversary corrupting the garbler and computational security against the evaluator, thus fall-back secure. We extend this to the multiparty setting for the "David and multiple Goliaths" case.

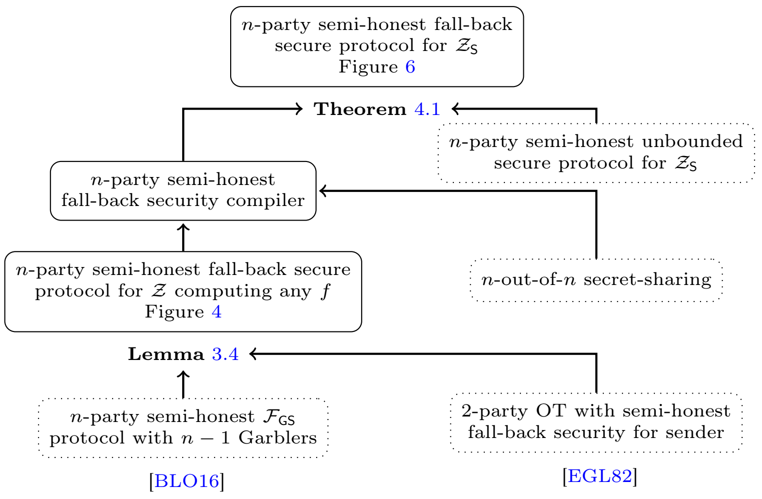
n-party semi-honest fall-back secure protocol for $\mathcal{Z}_\mathsf{S}$ Figure 6
Theorem 4.1
n-party semi-honest unbounded secure protocol for $\mathcal{Z}_\mathsf{S}$
n-party semi-honest fall-back security compiler
n-party semi-honest fall-back secure protocol for $\mathcal{Z}$ computing any f Figure 4
n-out-of-n secret-sharing
Lemma 3.4
n-party semi-honest $\mathcal{F}_\mathsf{GS}$ protocol with $n-1$ Garblers
2-party OT with semi-honest fall-back security for sender
[BLO16]
[EGL82]

Fig. 1. An overview of the building blocks of the construction in Section 4. $\mathcal{Z}$ is an adversary structure over n-parties having all subsets that don't include a designated party. Dotted boxes indicate primitives from prior work.

WARMUP: SOLVING THE DAVID AND MULTIPLE GOLIATHS PROBLEM. Let $\mathcal{P} = \{P_i\}_{i\in[n]}$ be a set of n-parties and let $\mathcal{Z}$ be an adversary structure containing all subsets of $\mathcal{P}$ that do not include P_n. We describe an n-party protocol that can compute any function with semi-honest fall-back security tolerating $\mathcal{Z}$. We remark that the work of [CDvdG87] solves this using a specific assumption and proceeds in a round-robin manner. Here, we describe a protocol that proceeds in $O(r)$-rounds, assuming an r-round OT protocol with one-sided statistical security, that is optimal in rounds and assumptions.

The protocol designates the first $n-1$ parties as garblers and P_n as the evaluator. First, the parties in $\{P_i\}_{i\in[n-1]}$ jointly compute a garbled circuit for the computed function using a distributed garbling protocol such as [BLO16] and hand it to P_n. This computation also produces certain 'masking bits' such that each party can XOR its inputs and masking bits to create a 'masked input' that reveals no information about the input itself. Each garbler then gives its masked input to P_n, which engages in multiple instances of oblivious transfer as the receiver, with each garbler being the sender. After these executions, P_n has received the input labels corresponding to all the masked inputs from each garbler. Finally, it uses these input labels to evaluate the garbling and sends the output to all the other parties.

The protocol is secure in the presence of a semi-honest unbounded adversary corrupting any subset of the garbling parties. This follows from the fact that a simulator for such an adversary would just need to honestly participate in the garbling, the oblivious transfer is secure in the presence of an unbounded sender and, since the adversary does not corrupt P_n, it never gets to see the masked inputs or input labels used for evaluation. The protocol is also secure in the presence of a semi-honest PPT adversary corrupting an arbitrary subset of $\mathcal{P}$. Hence any n-party function can be computed with semi-honest fall-back security tolerating $\mathcal{Z}$. Details for this protocol can be found in Section 3.1.

Compiling to Semi-Honest Fall-Back Security. The protocol above can be used as a building-block to construct a compiler that can convert any n-party protocol Π_{stat} that satisfies semi-honest security in the presence of an unbounded adversary tolerating an adversary structure $\mathcal{Z}_{\mathsf{S}}$, to a protocol Π_{in} that satisfies semi-honest fall-back security tolerating the same adversary structure $\mathcal{Z}_{\mathsf{S}}$.

The idea is to first parse Π_{stat} as a sequence of next-message functions, one for each party i in round j. For each such function, we define a functionality that receives as input from n parties corresponding to the *shares* of the view of party i, reconstructs this view, computes the next-message function, and shares the output among n parties using n-out-of-n secret-sharing.

Our compiler works by first having each party $P_i \in \mathcal{P}$ secret-share its inputs to Π_{stat}, to all other parties using n-out-of-n secret-sharing. Then for each next-message function for the i^{th} party in Π_{stat}, we use the protocol Π_{DG} from the "David and multiple Goliaths" setting outlined above to compute the corresponding functionality with party P_i as David, i.e. the designated evaluator. Once this has been executed for all next-message functions, all parties have a share of the output of every party in protocol Π_{stat}. Then, all parties (except P_i) give the shares of the i^{th} party in Π_{stat} to P_i. Each P_i reconstructs and obtains the output of the protocol.

We argue the security of the protocol in the presence of computational and unbounded adversaries. The computational setting follows directly as the next-message functions of Π_{stat} can be viewed as a functionality that computes the same functionality as the target functionality on secret shares and we rely on our warm-up protocol that offers security in the dishonest majority setting to implement each step in Π_{in}. Thus, essentially by composition, we obtain that every internal computation and message in the emulation of Π_{stat} remains hidden to any subset of the parties.

To argue security against a semi-honest unbounded adversary tolerating $\mathcal{Z}_{\mathsf{S}}$, we analyze the case when the adversary corrupts a subset $Z \in \mathcal{Z}_{\mathsf{S}}$. For every $i \notin Z$, it holds that the next-message computations of the virtual party S_i in Π_{in} has P_i as the Evaluator when executed via our warmup protocol. We have that the evaluator P_i is not corrupted by the adversary, and the security of the warmup protocol guarantees that the view of the virtual party S_i will be statistically hidden from Z (as Z does not include P_i). However, when $i \in Z$, an unbounded adversary controls P_i, and the warmup protocol is secure only if an unbounded adversary corrupts everyone but P_i. Since we are in the passive setting, we can conclude that each of these instances of the warmup protocol will proceed correctly. Still, an unbounded adversary can obtain the entire view of the virtual party S_i. This amounts to corrupting all parties indexed $i \in Z$ in the virtual protocol. We conclude from the statistical security of the virtual protocol Π_{in} against Z that the compiled protocol will also offer statistical security with respect to Z.

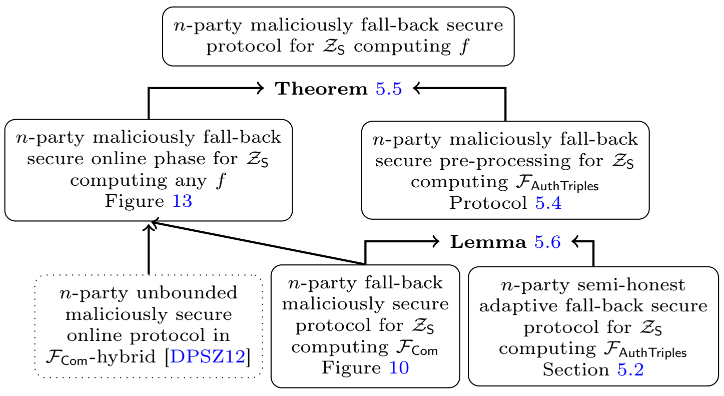
n-party maliciously fall-back secure protocol for $\mathcal{Z}_{\mathsf{S}}$ computing f
Theorem 5.5
n-party maliciously fall-back secure online phase for $\mathcal{Z}_{\mathsf{S}}$ computing any f Figure 13
n-party maliciously fall-back secure pre-processing for $\mathcal{Z}_{\mathsf{S}}$ computing $\mathcal{F}_{\mathsf{AuthTriples}}$ Protocol 5.4
Lemma 5.6
n-party unbounded maliciously secure online protocol in $\mathcal{F}_{\mathsf{Com}}$-hybrid [DPSZ12]
n-party fall-back maliciously secure protocol for $\mathcal{Z}_{\mathsf{S}}$ computing $\mathcal{F}_{\mathsf{Com}}$ Figure 10
n-party semi-honest adaptive fall-back secure protocol for $\mathcal{Z}_{\mathsf{S}}$ computing $\mathcal{F}_{\mathsf{AuthTriples}}$ Section 5.2

Fig. 2. An overview of the building blocks of the construction in Section 5. Dotted boxes indicate primitives from prior work.

Maliciously Secure Fall-Back Security. Next, we describe our protocol that achieves malicious fall-back security with respect to an adversary structure $\mathcal{Z}_{\mathsf{S}}$. The classical GMW paradigm is to rely on coin-tossing and distributed zero-knowledge proofs to compile a semi-honest secure protocol to a malicious one. Such an approach will require designing distributed commit-and-prove protocols with fall-back security with respect to $\mathcal{Z}_{\mathsf{S}}$ and typically results in a non-black-box construction in the underlying primitives. Instead, we will pursue another approach pioneered by Ishai, Prabhakaran, and Sahai [IPS08] that is based on player virtualization and allows compiling a "semi-honest" protocol to malicious in a modular way. This will have the benefit of not relying on distributed zero-knowledge proofs. We will use a slight variant of the compiler designed by Hazay et al. [HVW20], which is based on the IPS compiler.

In slightly more detail, we will rely on the offline-online paradigm refined in the SPDZ line of works [DPSZ12], where the parties first generate so-called *authenticated Beaver triples* in an offline input-independent phase and then consume them in an online phase. The main feature of this protocol relevant to our setting is that once the offline protocol is executed, the online protocol is unconditionally secure, assuming commitment schemes. Therefore, designing a malicious protocol with fall-back security reduces to designing a malicious fall-back secure protocol for the authenticated Beaver triples functionality and distributed commitment scheme with fall-back security. For our protocol to realize the authenticated triples functionality, we will need the following ingredients:

1. Fall-back secure extractable commitment scheme: We need a scheme where a single committer commits a value v to the $n-1$ parties that is unconditionally secure against $\mathcal{Z}_{\mathsf{S}}$, and computationally secure against arbitrary collusion. We will additionally need the commitment to be extractable.
2. Fall-back secure coin-tossing and coin-tossing-in-the-well protocols. We can use standard applications of $\mathcal{F}_{\mathsf{Com}}$ to obtain these protocols.
3. Semi-honest fall-back secure protocol for a simple functionality (closely related to the authenticated Beaver triples functionality).

Realizing $\mathcal{F}_{\mathsf{Com}}$. In the fall-back setting, the committer and (any one of) the receivers could be corrupted by an unbounded adversary (depending on $\mathcal{Z}_{\mathsf{S}}$). Recall that it is impossible to design a commitment scheme that is simultaneously unconditionally hiding and binding. Instead, we will rely on both a statistically hiding commitment $\mathsf{Com}_{\mathsf{SH}}$ and a statistically binding commitment $\mathsf{Com}_{\mathsf{SB}}$. Additionally, we will need a secret-sharing scheme with respect to adversary structure $\mathcal{Z}_{\mathsf{S}}$. First, we design a fall-back secure extractable commitment scheme and then compile it to a protocol that realizes $\mathcal{F}_{\mathsf{Com}}$.

The committer takes its value v, secret shares it to $v_1, \ldots, v_n$, and commits to v_i, with P_i being the receiver, using a statistically binding scheme, and commits to v using a statistically hiding commitment scheme. This scheme is hiding against a dishonest majority as both commitments are hiding against a computational adversary. This is statistically hiding against an unbounded adversary since it can only break the statistically-binding commitment scheme and learn secret shares of v that statistically hid it against corruption in $\mathcal{Z}_{\mathsf{S}}$. This commitment is binding in the dishonest majority setting because the committer uses a statistically hiding commitment scheme to commit v to every party. When an unbounded adversary corrupts in $\mathcal{Z}_{\mathsf{S}}$ (that includes the committer), the values of the secret shares committed to the honest parties and the statistically binding commitment statistically bind the value v. To make the scheme extractable, the committer uses computational zero-knowledge proof-of-knowledge (cZKPOK) to prove the knowledge of the values committed within $\mathsf{Com}_{\mathsf{SB}}$ and a statistical zero-knowledge argument-of-knowledge (sZKAOK) to prove the knowledge of the values committed via $\mathsf{Com}_{\mathsf{SH}}$. The simulator invokes the extractor of the sZKAOK in order to extract the inputs in the dishonest majority case and that of the cZKPOK to extract the input shares in the statistical case.

Next, to realize $\mathcal{F}_{\mathsf{Com}}$, we will rely on the lookahead trapdoor commitment scheme of [PW09]. Roughly speaking, in this commitment scheme, the committer, instead of committing to a bit σ, commits to κ 2×2-matrices

$$\begin{pmatrix} \eta_i & \eta_i \oplus \sigma \\ \eta_i & \eta_i \oplus \sigma \end{pmatrix}$$

where κ is the statistical security parameter and η_i's are chosen uniformly at random. Then the receiver provides a random κ-bit challenge to determine which columns in each of the κ matrices should be decommitted by the committer. The receiver accepts the commitment if the decommitment values in each column are equal. In the decommit phase, the committer picks a random row in each matrix and decommits the remaining value. This commitment can be made "equivocal" by having the receiver commit to its challenge before the committer's first commitment. This allows a simulator to "look ahead" the challenge and rewind to provide an equivocal commitment. Namely, each matrix will have identical values in the opened column and different bits in the unopened column so that in the decommitment phase, the simulator can choose the right opening depending on what value it needs to decommit to.

In our setting, we can essentially rely on this protocol, where in the first round, we have $n-1$ parties commit to a random string which it will decommit to in the third round, and the challenge will be the XOR of all strings. Next, the committer will use the weak-extractable commitment described above to commit to each entry of the matrices. The weak extractability suffices to extract the value, and the lookahead trapdoor property will help to equivocate.

Fall-back secure coin-tossing and coin-tossing in the well. Once we have a commitment scheme, we can rely on the standard mechanisms to design coin-tossing and coin-tossing-in-the-well schemes.

Putting it together. In the IPS-type compiler, the parties emulate a virtual "outer" malicious information theoretic MPC protocol among m parties (also denoted by servers) by maintaining each party's view in a secret-shared state and securely computing the next message functions via semi-honest "inner" protocol. In this work, we will rely on a vanilla outer protocol (such as BGW [BGW88]), whereas the inner protocol is instantiated using our semi-honest fall-back secure protocol. Since we are only implementing the authenticated Beaver triples functionality with a small depth, we can consider an outer protocol where the parties do not communicate with each other (i.e., all computations are local). This is not necessary for our compiler but simplifies our description and analysis.

On a high-level, our protocol proceeds in the following phases:

– Phase 1: The parties commit to a coin-tossing (by simply having each party commit to a random string) that will be opened in Phase 5.
– Phase 2: The parties secretly share their inputs for the m virtual parties and then commit to the shares using our fall-back extractable commitment scheme.
– Phase 3: The parties execute a coin-tossing-in-the-well protocol to generate randomness for the m virtual parties.
– Phase 4: All parties engage in m executions of the semi-honest fall-back secure protocol to emulate the m virtual parties.
– Phase 5: The parties open the coin-tossing and sample t of the m virtual parties where all parties open the randomness and inputs used for emulating those t parties.

The security proof follows a similar approach to [IPS08] and [HVW20]. We remark that (similar to [IPS08]), we will need the inner protocol to meet some adaptive security guarantees. We describe our protocol assuming this property but later argue that we can use an inner protocol without additional assumptions.

In more detail, in Phase 1, the simulator executes the coin-tossing honestly but extracts the outcome of the coin-toss and identifies the virtual servers whose views will be opened (referred to as watched executions) in the cut-and-choose phase (Phase 5 above). In Phase 2, the simulator extracts the adversary's inputs, sends the adversary's triples shares to the ideal functionality, and receives the offset as the output. Before discussing Phase 3, we mention that the idea for simulating Phase 4 is to have the simulator simulate the messages for the honest parties in the watched virtual parties honestly and rely on a simulation of the semi-honest fall-back secure protocol for the unwatched sessions. Since the outputs of the watched sessions are fixed, the simulator first generates the outputs for all of the remaining virtual parties in the outer protocol so that it is consistent with the output received from the ideal functionality and the output of the watched sessions. Next, it runs the semi-honest fall-back secure protocol simulator for the unwatched sessions to determine the randomness of the parties corrupted by the adversary in the unwatched sessions. Now, it proceeds with simulation, where in Phase 3, it will manipulate the coin-tossing-in-the-well protocol to fix the adversary's random tape in the unwatched sessions corresponding to the simulated views. Next, it will simulate Phase 5, hoping the adversary will not deviate in the unwatched sessions. If the adversary does indeed deviate in the emulation of one of the virtual parties, the simulator will invoke the adaptive simulation for that instance.

We will consider an outer protocol that is t private with $t/2$ sessions to watch. This will allow the simulator to generate inputs of the virtual parties (by simply sampling them uniformly at random) for up to $t/2$ more sessions where the adversary deviates. If it deviates in more than $t/2$ unwatched sessions, the simulator fails (where this will happen only with negligible probability due to the cut-and-choose check).

As with the analysis in [IPS08], proving security additionally requires the "inner" protocol, i.e. the semi-honest fall-back secure protocol to be adaptively secure where an adversary can eventually corrupt all parties. However, our semi-honest fall-back security does not satisfy this guarantee. In Section 5.5, we show how to modify our semi-honest protocol (without incurring any additional assumptions) to additionally satisfy a weaker adaptive security guarantee that will be sufficient to prove the security of our whole protocol.

2 Preliminaries

We denote by κ a computational security parameter and s a statistical security parameter that captures a statistical error of up to 2^{-s}. We assume $s \leq \kappa$.

Definition 2.1 *Two probability ensembles $X = \{X_n\}_{n \in \mathbb{N}}$ and $Y = \{Y_n\}_{n \in \mathbb{N}}$ defined over a finite domain D are* ***statistically indistinguishable****, denoted $X \overset{s}{\equiv} Y$, if every positive polynomial $p(\cdot)$ and all sufficiently large n's,*

$$\Delta(X_n, Y_n) < \frac{1}{p(n)}$$

where,

$$\Delta(X_n, Y_n) = \frac{1}{2} \cdot \sum_{\alpha \in D} |\Pr[X_n = \alpha] - \Pr[Y_n = \alpha]|$$

Definition 2.2 *Two probability ensembles $X = \{X_n\}_{n \in \mathbb{N}}$ and $Y = \{Y_n\}_{n \in \mathbb{N}}$ are* ***computationally indistinguishable****, denoted $X \overset{c}{\equiv} Y$, if for every PPT distinguisher D, every positive polynomial $p(\cdot)$ and all sufficiently large n's,*

$$|\Pr[D(X_n, 1^n) = 1] - \Pr[D(Y_n, 1^n) = 1]| < \frac{1}{p(n)}$$

Secure Multiparty Computation. Let $\mathcal{P} = \{P_i\}_{i \in [n]}$ be a set of n parties. MPC involves the computation of a random process that maps a tuple of inputs to a tuple of outputs (one for each party). This random process is referred to as a functionality $f : \{0,1\}^* \times \cdots \times \{0,1\}^* \to \{0,1\}^* \times \cdots \times \{0,1\}^*$ where $f = (f_1, \ldots, f_n)$. Each party P_i has input x_i. For simplicity and without loss of generality, we work over functionalities for which $x_i \in \{0,1\}$. For every tuple of inputs $\vec{x} = (x_i, \ldots, x_n)$, the vector output by f is a random variable $(f_1(\vec{x}), \ldots, f_n(\vec{x}))$ ranging over tuples of strings where P_i receives $f_i(\vec{x})$. The following notation describes a functionality:

$$(x_1, \ldots, x_n) \mapsto (f_1(\vec{x}), \ldots, f_n(\vec{x}))$$

We prove the security of our MPC protocols in the presence of semi-honest and malicious adversaries in the dishonest majority setting.

In the semi-honest setting, an adversary can control a subset of the participating parties and always follows the protocol specification but may try to learn additional information from the transcript of messages received and its internal state. Let $f = (f_1, \ldots, f_n)$ be a multiparty functionality and Π be an n-party protocol computing f in t rounds. The view of party P_i in an execution of Π on inputs $\vec{x} = (x_1, \ldots, x_n)$ is,

$$\mathbf{View}_{\Pi,i}(\kappa, \vec{x}) = (\kappa, x_i, r_i, \{m^j_{k\to i}\}_{j\in[t],k\neq i\in[n]})$$

where κ is the security parameter, r_i is the content of P_i's internal random tape, and each $m^j_{k\to i}$ represents the message received from party P_k in round j. The output of P_i in an execution of Π on $\vec{x}$ is denoted $\mathsf{output}_{\Pi,i}(\vec{x})$ and can be computed from $\mathbf{View}_{\Pi,i}(\kappa, \vec{x})$. The set of corrupted parties is denoted by $Z \subset \mathcal{P}$ and the set of honest parties by $\overline{Z}$. We extend the view notation to capture any subset of parties, denoting by $\mathbf{View}_{\Pi,T}(\kappa, \vec{x})$ the joint view of all parties in $T \subseteq \mathcal{P}$ on $(\kappa, \vec{x})$.

Definition 2.3 *Let f and Π be as above, and let $\mathcal{Z}_{\mathsf{C}} \subseteq 2^{\mathcal{P}}$ be an adversary structure. The protocol Π is said to securely compute f in the presence of a **PPT semi-honest adversary** tolerating adversary structure $\mathcal{Z}_{\mathsf{C}}$ if there exists a PPT simulator Sim such that for every $Z \in \mathcal{Z}_{\mathsf{C}}$,*

$$\left\{\mathsf{Sim}(1^\kappa, Z, \{x_i, f_i(\vec{x})\}_{i\in Z}), \{f_i(\vec{x})\}_{i\notin Z}\right\}_{\kappa\in\mathbb{N},\vec{x}\in\{0,1\}^*}$$
$$\overset{\mathrm{c}}{\equiv}\left\{\mathbf{View}_{\Pi,Z}(\kappa, \vec{x}), \mathsf{output}_{\Pi,\overline{Z}}(\kappa, \vec{x})\right\}_{\kappa\in\mathbb{N},\vec{x}\in\{0,1\}^*}$$

where κ is the computational security parameter. The distribution is considered over the randomness of the simulator and random tapes of parties in the protocol.

Definition 2.4 *Let f and Π be as above, and let $\mathcal{Z}_{\mathsf{S}} \subseteq 2^{\mathcal{P}}$ be an adversary structure. The protocol Π is said to securely compute f in the presence of an **unbounded semi-honest adversary** tolerating adversary structure $\mathcal{Z}_{\mathsf{S}}$ if there exists a PPT simulator Sim such that for every $Z \in \mathcal{Z}_{\mathsf{S}}$,*

$$\left\{\mathsf{Sim}(1^s, Z, \{x_i, f_i(\vec{x})\}_{i\in Z}), \{f_i(\vec{x})\}_{i\notin Z}\right\}_{s\in\mathbb{N},\vec{x}\in\{0,1\}^*}$$
$$\overset{\mathrm{s}}{\equiv}\left\{\mathbf{View}_{\Pi,Z}(s, \vec{x}), \mathsf{output}_{\Pi,\overline{Z}}(s, \vec{x})\right\}_{s\in\mathbb{N},\vec{x}\in\{0,1\}^*}$$

where s is the statistical security parameter. The distribution is considered over the simulator's randomness and the parties' random tapes in the protocol.

In the malicious setting, an adversary controlling a subset of the participating parties can have them deviate arbitrarily from the protocol specification. Security in this setting is defined by a comparison between an execution of the functionality in an ideal model and an execution of the protocol in the real model.

Execution in the ideal model. An ideal execution is an interaction between all the parties in $\mathcal{P}$ and a trusted party $\mathcal{F}$, wherein the parties submit their input to the trusted party that computes the output and returns it. An honest party receives input for the computation and forwards it to the trusted party. In contrast, a corrupt party can replace its input with an arbitrary value of the same length. On receiving these, $\mathcal{F}$ computes the output and first sends the outputs of the corrupt parties to the adversary. The adversary then decides whether the honest parties will receive their outputs from $\mathcal{F}$ or an abort symbol $\perp$. For a multiparty functionality $f = (f_1, \ldots, f_n)$, let $\mathcal{A}$ be a non-uniform probabilistic machine and $Z \subset \mathcal{P}$ be the set of corrupted parties. Then the ideal execution of f on inputs $(\kappa, \vec{x})$, auxiliary input z to $\mathcal{A}$ and security parameter κ is defined as the output tuple of the honest parties and the adversary from this execution and is denoted as $\mathbf{IDEAL}_{f,\mathcal{A}(z),Z}(\kappa, \vec{x})$.

Execution in the real model. This is an interaction directly among the parties in $\mathcal{P}$ wherein the honest parties follow the instructions in the protocol Π. The adversary $\mathcal{A}$ sends all messages in place of the corrupted parties and may follow any arbitrary strategy. Letting Π be a protocol that computes f as above, $\mathcal{A}$ be a non-uniform probabilistic machine, and let $Z \subset \mathcal{P}$ be the set of corrupted parties, the real execution of Π on inputs $(\kappa, \vec{x})$, auxiliary input z to $\mathcal{A}$ and security parameter κ is defined as the output tuple of the honest parties and the adversary from this execution and is denoted as $\mathbf{REAL}_{\Pi,\mathcal{A}(z),Z}(\kappa, \vec{x})$.

Definition 2.5 *Let f and Π be as above, and let $\mathcal{Z}_{\mathsf{C}} \subseteq 2^{\mathcal{P}}$ be an adversary structure. The protocol Π is said to securely compute f with abort in the presence of* ***PPT malicious adversaries*** *tolerating adversary structure $\mathcal{Z}_{\mathsf{C}}$ if there exists a PPT adversary Sim in the ideal model such that for every PPT $\mathcal{A}$ in the real model and $Z \in \mathcal{Z}_{\mathsf{C}}$,*

$$\left\{\mathbf{IDEAL}_{f,\mathsf{Sim}(z),Z}(\kappa, \vec{x})\right\}_{\kappa\in\mathbb{N},\vec{x},z\in\{0,1\}^*} \overset{\mathrm{c}}{\equiv} \left\{\mathbf{REAL}_{\Pi,\mathcal{A}(z),Z}(\kappa, \vec{x})\right\}_{\kappa\in\mathbb{N},\vec{x},z\in\{0,1\}^*}$$

where κ is the computational security parameter. The distribution is considered over the adversaries' randomness and all the parties' random tapes.

Definition 2.6 *Let f and Π be as above, and let $\mathcal{Z}_{\mathsf{S}} \subseteq 2^{\mathcal{P}}$ be an adversary structure. The protocol Π is said to securely compute f with abort in the presence of* ***unbounded malicious adversaries*** *tolerating adversary structure $\mathcal{Z}_{\mathsf{S}}$ if there exists an adversary Sim in the ideal model such that for every computationally unbounded $\mathcal{A}$ in the real model and every $Z \in \mathcal{Z}_{\mathsf{S}}$,*

$$\left\{\mathbf{IDEAL}_{f,\mathsf{Sim}(z),Z}(s, \vec{x})\right\}_{s\in\mathbb{N},\vec{x},z\in\{0,1\}^*} \overset{\mathrm{s}}{\equiv} \left\{\mathbf{REAL}_{\Pi,\mathcal{A}(z),Z}(s, \vec{x})\right\}_{s\in\mathbb{N},\vec{x},z\in\{0,1\}^*}$$

where s is the computational security parameter. The distribution is considered over the adversaries' randomness and all the parties' random tapes.

Oblivious Transfer. This is a 2-party functionality $\mathcal{F}_{\mathsf{OT}}$ between a sender S and a receiver R of the form:

$$\mathsf{OT} : ((s_0, s_1), b) \mapsto (\bot, s_b)$$

S has two strings $s_0, s_1 \in \{0,1\}^\ell$ as inputs and R has a choice bit $b \in \{0,1\}$. The functionality gives s_b as output to R. In our construction, we require a sub-protocol Π_{OT} that computes $\mathcal{F}_{\mathsf{OT}}$ and is secure even when the sender is corrupted by a computationally unbounded adversary (and a receiver is corrupted by a PPT adversary). We require such a protocol in the semi-honest [EGL82] setting.

Garbling Schemes. For a function $f : \{0,1\}^n \to \{0,1\}^m$, let $\mathbf{C}$ be its circuit representation. A garbling scheme [BHR12] is a cryptographic primitive that has the following structure:

Definition 2.7 (Garbling Scheme) *Let $f : \{0,1\}^n \to \{0,1\}^m$ be a function. Let κ be a computational security parameter. A garbling scheme $\mathsf{GS} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev})$ consists of four polynomial-time algorithms:*

- $(F, e, d) \leftarrow \mathsf{Gb}(1^\kappa, f)$*: returns a garbling F, input encoding set e, and output decoding set d.*
- $X \leftarrow \mathsf{En}(e, x)$*: returns the encoding X for function input x.*
- $Y \leftarrow \mathsf{Ev}(F, X)$*: returns the output labels Y by evaluating F on X.*
- $\{\bot, y\} \leftarrow \mathsf{De}(Y, d)$*: returns either the failure symbol $\bot$ or a value $y = f(x)$.*

These algorithms must satisfy the following properties:

- ***Correctness**: For every κ, function f and input x,*

$$\Pr[y = f(x) \ : (F, e, d) \leftarrow \mathsf{Gb}(f),\ X = \mathsf{En}(e, x),\ Y = \mathsf{Ev}(F, X),\ y = \mathsf{De}(d, Y)] = 1$$

- ***Privacy**: Let Algorithm 1 denote the actions of the challenger $\mathcal{C}$ in a simulation game. For all PPT adversaries $\mathcal{A}$ there exists a PPT simulator Sim such that for all f and x, for a negligible function μ, $\mathcal{A}$'s advantage is,*

$$\mathsf{Adv}(\kappa) = \left| \Pr[\mathcal{A}(f, x, F, X, d) = b] - \frac{1}{2} \right| < \mu(\kappa)$$

Algorithm 1 Privacy

```
1: proc $\mathsf{Challenger}(f, x)$
2:     $b \leftarrow \{0,1\}$
3:     if $b == 0$,
4:         $(F, e, d) \leftarrow \mathsf{Gb}(1^\kappa, f)$
5:         $X = \mathsf{En}(e, x)$
6:     if $b == 1$,
7:         $(F, X, d) \leftarrow \mathsf{Sim}(1^\kappa, f(x), f)$
8:     Return $(F, X, d)$
```

Multi-Party Garbling. For a function $f : \{0,1\}^n \to \{0,1\}^m$, let $\mathbf{C}$ be its circuit representation. Let q be the number of gates in $\mathbf{C}$. Each gate $g \in [q]$ is defined by a gate functionality $f_g \in \{\mathsf{AND}, \mathsf{XOR}\}$, two input wires A, B and an output wire g where, $A, B, g \in [n+q]$ and topological ordering holds: $A, B < g$.

A multi-party garbling protocol is one in which a set of parties use their combined randomness to create a garbled representation of a function. In such a protocol, it should hold that even when all-but-one of the garblers collude, no information about the randomness contributed by the non-colluding party is revealed beyond what can be deduced from the final garbling. There exist protocols in the literature [BMR90, BLO16] that compute this for an n-party garbling protocol in the presence of a semi-honest corruption of any $n-1$ parties. In our construction, we require a multi-party garbling functionality that operates as in Figure 3. Our protocols use a multi-party garbling protocol that implements the functionality $\mathcal{F}_{\mathsf{GS}}$ (Figure 3) as a sub-protocol where $\mathsf{N} = n - 1$ parties participate in the joint garbling process. We require one such protocol that is secure in the presence of PPT semi-honest adversaries.

Zero-Knowledge Proofs. Zero-Knowledge Proofs of Knowledge are an interaction between a prover P and a verifier V such that the interaction convinces V that for a public value x, P has a secret witness w such that $(x, w) \in R$ for a public relation R but the interaction reveals nothing beyond that fact. We use two variants of zero-knowledge proofs. The first is a statistical zero-knowledge argument of knowledge where the prover is computationally bounded, but the protocol is zero-knowledge even in the presence of an unbounded verifier.

Definition 2.8 *Let μ be a negligible function, and for computational security parameter κ, let $\epsilon = \mu(\kappa)$. A pair of machines $\mathsf{SZKAoK} = (P, V)$ is a* ***statistical zero-knowledge argument of knowledge*** *for a relation R if for every V,*

– ***Perfect Completeness:*** *For every $(x, w) \in R$,*

$$\Pr[\langle P(w), V\rangle(x) = 1] \geq 1 - \epsilon$$

– ***ϵ-Soundness:*** *For every x such that $\nexists w$ for which $(x, w) \in R$, and every malicious PPT P^*,*

$$\Pr[\langle P^*, V\rangle(x) = 1] \leq \epsilon$$

– ***Statistical Zero-Knowledge:*** *For every unbounded V^* there is a PPT S^*,*

$$\{\langle P(w), V^*\rangle(x)\}_{(x,w)\in R} \overset{\mathrm{s}}{\equiv} \{S^*(x)\}_{(x,w)\in R}$$

– ***Knowledge Soundness:*** *There exists a PPT E^* such that for every PPT prover P^*, if $\Pr[\langle P^*, V\rangle(x) = 1] \geq 1 - \epsilon$ then,*

$$\Pr[\langle P^*, E^*\rangle(x) = w] \geq 1 - \epsilon - \sigma$$

where σ (negligible in κ) is the knowledge error.

Distributed Garbling

Setting: Let f be a function with input $x = \{x_i\}_{i \in [n]}$ and circuit representation $\mathbf{C}$. Let $\kappa = \ell$ be a security parameter and $\mathcal{P}_{\mathsf{N}} = \{P_i\}_{i \in [\mathsf{N}]}$ be the set of garblers where $\mathcal{P}_{\mathsf{N}} \subseteq \mathcal{P} = \{P_i\}_{i \in [n]}$. Let $\mathsf{PRF} : \{0,1\}^{\kappa} \times \{0,1\}^{\kappa} \times \{0,1\}^{\log q + \log \mathsf{N}} \rightarrow \{0,1\}^{\kappa+1}$ be a pseudo-random function. The functionality $\mathcal{F}_{\mathsf{GS}}$ operates as follows:

- **Inputs:** Each party $P_i \in \mathcal{P}_{\mathsf{N}}$ has the following inputs:
  - for each input wire $w \in [n]$, if $w = i$ or $w \notin [\mathsf{N}]$, then $\lambda_w^i \leftarrow \{0,1\}$; else $\lambda_w^i = 0$
  - for each gate output wire $w \in [q]$, sample $\lambda_w^i \leftarrow \{0,1\}$
  - for each wire $w \in [n+q]$, sample labels $k_{w,0}^i, k_{w,1}^i \leftarrow \{0,1\}^{\ell}$
  - for all $j \in [\mathsf{N}], (a,b) \in \{0,1\}^2$ and gate $g \in [q]$ with inputs A and B,

  $$F_{a,b}^{i,j,g} = \mathsf{PRF}_{k_{A,a}^i, k_{B,b}^i}(g||j)$$

  $\mathcal{F}_{\mathsf{GS}}$ gets from P_i:

  $$\{\lambda_w^i, k_{w,0}^i, k_{w,1}^i\}_{w \in [n+q]}, \{F_{a,b}^{i,j,g}\}_{g \in [q], j \in [\mathsf{N}], (a,b) \in \{0,1\}^2}$$

  Each party $P_i \in \mathcal{P} - \mathcal{P}_{\mathsf{N}}$ does not provide any input.
- **Computation:** The functionality $\mathcal{F}_{\mathsf{GS}}$ computes for all gate $g \in [q]$ with functionality f_g, $\forall (a,b) \in \{0,1\}^2, \forall j \in [\mathsf{N}]$:

  $$G_{a,b}^j = \left(\bigoplus_{i=1}^{\mathsf{N}} F_{a,b}^{i,j,g} \right) \oplus ((\lambda_w \oplus f_g(a,b))||k_{w,f_g(a,b)}^j)$$

- **Outputs:** $\mathcal{F}_{\mathsf{GS}}$ gives to all parties $P_i \in \mathcal{P}$:
  - The garbling
    $$\{G_{a,b}^j\}_{j \in [\mathsf{N}], g \in [q], (a,b) \in \{0,1\}^2}$$
  - For each circuit output wire $w \in [n+q-m, n+q]$, $\lambda_w = \bigoplus_{i=1}^{\mathsf{N}} \lambda_w^i$

  Each party $P_i \in \mathcal{P} - \mathcal{P}_{\mathsf{N}}$ gets for input wire $w = i$, $\lambda_w = \bigoplus_{i=1}^{\mathsf{N}} \lambda_w^i$
  Each garbler $P_i \in \mathcal{P}_{\mathsf{N}}$ sets for input wire $w = i$, $\lambda_w = \lambda_w^i$

Fig. 3. Distributed Garbling

The second variant is a computational zero-knowledge proof of knowledge protocol between an unbounded prover and a computationally bounded verifier.

Definition 2.9 *Let μ be a negligible function, and for computational security parameter κ, let $\epsilon = \mu(\kappa)$. The tuple* $\mathsf{ZKPoK} = (P, V)$ *is a* ***computational zero-knowledge proof of knowledge*** *for a relation R if for every PPT V,*

- ***Perfect Completeness:*** *For every $(x, w) \in R$,*

$$\Pr[\langle P(w), V \rangle(x) = 1] \geq 1 - \epsilon$$

- ***ϵ-Soundness:*** *For every x such that $\nexists w$ for which $(x, w) \in R$, and every malicious P^*,*

$$\Pr[\langle P^*, V\rangle(x) = 1] \leq \epsilon$$

- ***Computational Zero-Knowledge:*** *For every PPT V^* there is a PPT S^*,*

$$\{\langle P(w), V^*\rangle(x)\}_{(x,w)\in R} \overset{c}{\equiv} \{S^*(x)\}_{(x,w)\in R}$$

- ***Knowledge Soundness:*** *There exists a PPT E^* such that for every prover P^*, if $\Pr[\langle P^*, V\rangle(x) = 1] \geq 1 - \epsilon$ then,*

$$\Pr[\langle P^*, E^*\rangle(x) = w] \geq 1 - \epsilon - \sigma$$

where σ (negligible in κ) is the knowledge error.

Commitment Schemes. A commitment scheme Com is a 2-party protocol between a committer C with an input x and a receiver R in which C samples randomness r and creates a 'commitment' to the value x. This commitment should be hiding: no information about x should be revealed. At a later point in time, C can send a 'decommit' to message to R showing that the value committed to was indeed x. This property by which C cannot deceive R into believing that the value committed to was something different $x' \neq x$ is called the binding property of the commitment. In our protocols, we use statistically binding commitments to ensure that an unbounded malicious adversary cannot deviate from the protocol.

Definition 2.10 $\mathsf{Com}_{\mathsf{SB}}(m; r)$ *is a **Statistically Binding Commitment** for messages $m \in \{0,1\}^\ell$ if it satisfies:*

- ***Statistical Binding:*** *For all messages $m_0, m_1 \in \{0,1\}^\ell$ it holds that,*

$$\{\mathsf{Com}_{\mathsf{SB}}(m_0; r)\}_{r\in\{0,1\}^*} \cap \{\mathsf{Com}_{\mathsf{SB}}(m_1; s)\}_{s\in\{0,1\}^*} = \phi$$

- ***Computationally Hiding:*** *For all choice of $m_0, m_1 \in \{0,1\}^\ell$ it holds that,*

$$\{\mathsf{Com}_{\mathsf{SB}}(m_0; r)\}_{r\in\{0,1\}^*} \overset{c}{\equiv} \{\mathsf{Com}_{\mathsf{SB}}(m_1; s)\}_{s\in\{0,1\}^*}$$

- ***Decommitment*** *to a commitment τ is (m, r) such that $\tau = \mathsf{Com}_{\mathsf{SB}}(m; r)$*

Since the binding and hiding properties of commitments are contradicting, a commitment scheme cannot be both statistically binding and hiding. But there exist commitment schemes that are either statistically hiding and computationally binding, or statistically binding and computationally hiding. In our work, we require both kinds of commitment schemes.

Definition 2.11 *A non-interactive **Statistically Hiding Commitment** scheme for a message space $\{0,1\}^\ell$ is a PPT algorithm $\mathsf{Com}_{\mathsf{SH}}$ satisfying:*

– ***Computationally Binding:*** *$\forall$ PPT adversaries $\mathcal{A}$ there exists a negligible function μ such that,*

$$\Pr[\mathcal{A}(1^\kappa) \to (c, m_0, r_0, m_1, r_1) | \mathsf{Com}_{\mathsf{SH}}(m_0; r_0) = c; \; \mathsf{Com}_{\mathsf{SH}}(m_1; r_1) = c; \; m_0, m_1 \in \{0,1\}^\ell \quad] < \mu(\kappa)$$

– ***Statistically Hiding:*** *For all choice of $m_0, m_1 \in \{0,1\}^\ell$ it holds that,*

$$\{\mathsf{Com}_{\mathsf{SH}}(m_0; r)\}_{r\in\{0,1\}^*} \overset{\mathrm{s}}{\equiv} \{\mathsf{Com}_{\mathsf{SH}}(m_1; s)\}_{s\in\{0,1\}^*}$$

– ***Decommitment*** *to a commitment c is (m, r) such that $c = \mathsf{Com}_{\mathsf{SH}}(m; r)$*

Secret Sharing. A secret-sharing scheme is a tuple of PPT functions $\mathsf{SS} = (\mathsf{Share}, \mathsf{Recon})$. It is defined with security against an adversary structure $\mathcal{Z} \subset 2^{\mathcal{P}}$ over n parties. For a message m, the function $\mathsf{Share}(m)$ returns n values $\{[m]_i\}_{i\in[n]}$ such that for any set $Z \in \mathcal{Z}$, the set of shares $\{[m]_i\}_{P_i\in Z}$ reveal no information about m, even to a computationally unbounded adversary. However, for any set $Z \notin \mathcal{Z}$, the set of shares $\{[m]_i\}_{P_i\in Z}$ reveal the message: $m = \mathsf{Recon}(\{[m]_i\}_{P_i\in Z})$.

Let $\mathsf{SS}_{n,n} = (\mathsf{Share}_{n,n}, \mathsf{Recon}_{n,n})$ be an n-out-of-n secret-sharing scheme. That is, for any message m, the tuple $\{[m]_i\}_{i\in[n]} \leftarrow \mathsf{Share}_{n,n}(m)$ is such that any proper subset of this set of shares reveal no information about m to an unbounded adversary. However, it holds that $m = \mathsf{Recon}_{n,n}(\{[m]_i\}_{i\in[n]})$. Such a scheme exists [Sha79] and it is also additively homomorphic:

$$\begin{aligned} \{[m_1]_i\}_{i\in[n]} &\leftarrow \mathsf{Share}_{n,n}(m_1) \\ \{[m_2]_i\}_{i\in[n]} &\leftarrow \mathsf{Share}_{n,n}(m_2) \\ \forall i \in [n], [m_1 + m_2]_i &= [m_1]_i + [m_2]_i \text{ s.t.} \\ m_1 + m_2 &= \mathsf{Recon}_{n,n}(\{[m_1 + m_2]_i\}_{i\in[n]}) \end{aligned}$$

Similarly, we denote by $\mathsf{SS}_{t,n} = (\mathsf{Share}_{t,n}, \mathsf{Recon}_{t,n})$ an t-out-of-n threshold secret-sharing scheme. That is, for any message m, the tuple $\{[m]_i\}_{i\in[n]} \leftarrow \mathsf{Share}_{t,n}(m)$ is such that any subset of t elements of this set of shares reveal no information about m to an unbounded adversary. However, for any set $Z \subseteq \mathcal{P}$ where $|Z| > t$, it holds that $m = \mathsf{Recon}_{t,n}(\{[m]_i\}_{P_i\in Z})$. Such a scheme exists for any $t < n$ [Sha79] and it is additively homomorphic. It is also homomorphic over multiplication at the cost of increasing the threshold:

$$\begin{aligned} \{[m_1]_i\}_{i\in[n]} &\leftarrow \mathsf{Share}_{t,n}(m_1) \\ \{[m_2]_i\}_{i\in[n]} &\leftarrow \mathsf{Share}_{t,n}(m_2) \\ \forall i \in [n], [m_1 * m_2]_i &= [m_1]_i * [m_2]_i \text{ s.t. } \forall Z \in \mathcal{P}, |Z| > 2t, \\ m_1 * m_2 &= \mathsf{Recon}_{2t,n}(\{[m_1 * m_2]_i\}_{i\in Z}) \end{aligned}$$

Finally, in its complete generality, for an adversary structure $\mathcal{Z} \subset 2^{\mathcal{P}}$, we denote by $\mathsf{SS}_{\mathcal{Z}} = (\mathsf{Share}_{\mathcal{Z}}, \mathsf{Recon}_{\mathcal{Z}})$ a secret-sharing scheme that tolerates $\mathcal{Z}$. That is, for any message m, the tuple $\{[m]_i\}_{i\in[n]} \leftarrow \mathsf{Share}_{\mathcal{Z}}(m)$ is such that for any $Z \in \mathcal{Z}$, the elements of this set $\{[m]_i\}_{P_i \in Z}$ reveal no information about m to an unbounded adversary. However, for any set $Z \notin \mathcal{Z}$ it holds that $m = \mathsf{Recon}_{\mathcal{Z}}(\{[m]_i\}_{P_i\in Z})$. Such a scheme exists for any $\mathcal{Z}_{\mathsf{S}}$ [BK95] that is an adversary structure under which there exist MPC protocols realizing general functionalities in the presence of a semi-honest unbounded adversary.

3 MPC with Fall-Back Security

We define a new notion of multi-party computation: MPC with fall-back security. Informally, a multi-party protocol satisfies fall-back security if it is secure when certain subsets of parties are corrupted by a computationally unbounded adversary. However, if more parties than that are corrupted, computational security is still guaranteed.

Definition 3.1 (Semi-Honest Fall-Back Security) *For a set of parties $\mathcal{P} = \{P_i\}_{i\in[n]}$ where P_i has input x_i, let $\Pi = \langle \mathcal{P} \rangle$ be an n-party protocol computing a function f with input $\vec{x} = \{x_i\}_{i\in[n]}$. Let $\mathcal{Z}_{\mathsf{C}} = 2^{\mathcal{P}}$ and $\mathcal{Z}_{\mathsf{S}} \subseteq \mathcal{Z}_{\mathsf{C}}$ be adversary structures for PPT and unbounded semi-honest adversaries, respectively. The protocol Π satisfies **fall-back security in the semi-honest setting tolerating** $\mathcal{Z}_{\mathsf{S}}$ if there exists a PPT Sim such that for every $Z \in \mathcal{Z}_{\mathsf{C}}$,*

$$\left\{\mathsf{Sim}(1^\kappa, Z, \{x_i, f_i(\vec{x})\}_{i\in Z}), \{f_i(\vec{x})\}_{i\notin Z}\right\}_{\kappa\in\mathbb{N}, \vec{x}\in\{0,1\}^*}$$
$$\overset{\mathrm{c}}{\equiv}\left\{\mathbf{View}_{\Pi,Z}(\kappa,\vec{x}), \mathsf{output}_{\Pi,\overline{Z}}(\kappa,\vec{x})\right\}_{\kappa\in\mathbb{N},\vec{x}\in\{0,1\}^*}$$

where κ is the computational security parameter; and there exists a PPT simulator Sim such that for every $Z \in \mathcal{Z}_{\mathsf{S}}$,

$$\left\{\mathsf{Sim}(1^s, Z, \{x_i, f_i(\vec{x})\}_{i\in Z}), \{f_i(\vec{x})\}_{i\notin Z}\right\}_{s\in\mathbb{N}, \vec{x}\in\{0,1\}^*}$$
$$\overset{\mathrm{s}}{\equiv}\left\{\mathbf{View}_{\Pi,Z}(s,\vec{x}), \mathsf{output}_{\Pi,\overline{Z}}(s,\vec{x})\right\}_{s\in\mathbb{N},\vec{x}\in\{0,1\}^*}$$

where s is the statistical security parameter. The distribution is considered over the randomness of the simulator and all random tapes of parties in the protocol.

Malicious Fall-Back Security. Fall-back security can be similarly defined also in the malicious setting.

Definition 3.2 (Malicious Fall-Back Security) *For a set of parties $\mathcal{P} = \{P_i\}_{i \in [n]}$ where P_i has input x_i, let $\Pi = \langle \mathcal{P} \rangle$ be an n-party protocol computing a function f with input $\vec{x} = \{x_i\}_{i \in [n]}$. Let $\mathcal{Z}_\mathsf{C} = 2^{\mathcal{P}}$ and $\mathcal{Z}_\mathsf{S} \subseteq \mathcal{Z}_\mathsf{C}$ be adversary structures for PPT and computationally unbounded malicious adversaries, respectively. The protocol Π satisfies **fall-back security in the malicious setting tolerating** $\mathcal{Z}_\mathsf{S}$ if there exists a PPT adversary* Sim *in the ideal model such that for every PPT $\mathcal{A}$ in the real model and every $Z \in \mathcal{Z}_\mathsf{C}$,*

$$\left\{\mathbf{IDEAL}_{f,\mathsf{Sim}(z),Z}(\kappa, \vec{x})\right\}_{\kappa \in \mathbb{N}, \vec{x}, z \in \{0,1\}^*} \overset{\mathrm{c}}{\equiv} \left\{\mathbf{REAL}_{\Pi,\mathcal{A}(z),Z}(\kappa, \vec{x})\right\}_{\kappa \in \mathbb{N}, \vec{x}, z \in \{0,1\}^*}$$

where κ is the computational security parameter; and there exists an adversary Sim *in the ideal model such that for every computationally unbounded $\mathcal{A}$ in the real model and every $Z \in \mathcal{Z}_\mathsf{S}$,*

$$\left\{\mathbf{IDEAL}_{f,\mathsf{Sim}(z),Z}(s, \vec{x})\right\}_{s \in \mathbb{N}, \vec{x}, z \in \{0,1\}^*} \overset{\mathrm{s}}{\equiv} \left\{\mathbf{REAL}_{\Pi,\mathcal{A}(z),Z}(s, \vec{x})\right\}_{s \in \mathbb{N}, \vec{x}, z \in \{0,1\}^*}$$

where s is the statistical security parameter. The distribution is considered over the randomness of the simulator and all random tapes of parties in the protocol.

3.1 Example Protocol with Semi-Honest Fall-Back Security

The protocol in Figure 4 is an n-party protocol Π in the $(\mathcal{F}_{OT}, \mathcal{F}_\mathsf{GS})$-hybrid that can compute any function f. In the protocol, out of the set of parties $\mathcal{P} = \{P_i\}_{i \in [n]}$, the parties $\{P_i\}_{i \in [n-1]}$ are designated the garblers and P_n is the evaluator. The garblers collectively create a garbled circuit F for the function f. This garbled circuit F and the output decoding information d are given to P_n. The evaluator P_n also receives from each garbling party P_i a 'masked value' Λ_i that hides its private input x_i. It then engages as a receiver in an OT protocol with each garbling party P_i as the sender and receives the active labels corresponding to all the masked values. P_n, with all the active labels, evaluates the garbled circuit to derive the output $f(\vec{x})$. It then sends $f(\vec{x})$ to all the other parties.

Figure 4 is a protocol with fall-back security against an unbounded adversary corrupting any subset of $\mathcal{P}_\mathsf{N}$ and a PPT adversary corrupting any subset of parties in $\mathcal{P}$. This also holds in the plain model when $\mathcal{F}_\mathsf{OT}$ is replaced with a semi-honest secure OT protocol that is secure against an unbounded sender and PPT receiver [EGL82]; and when $\mathcal{F}_\mathsf{GS}$ is replaced by a multiparty garbling protocol [BLO16] that involves only the parties in $\mathcal{P}_\mathsf{N}$ for garbling the circuit.

Remark 3.3 (On the necessity of OT.) *One may initially assume that, as in a protocol secure in the presence of PPT adversaries, once a garbler $P_i \in \mathcal{P}_\mathsf{N}$ publishes Λ_i, then every other garbler can publish its labels corresponding to these values without requiring OT. However, this violates privacy in the case of an unbounded adversary corrupting a strict subset of the garblers. Here, there exists an honest garbler participating in the Garbling Phase. However, since the*

Semi-honest Fall-Back Secure Protocol

Setting: Let $\mathcal{P} = \{P_i\}_{i\in[n]}$ be such that P_i has input x_i. Let f be a function, $\vec{x} = \{x_i\}_{i\in[n]}$ be its input and the protocol outputs $f(\vec{x})$ to all parties in $\mathcal{P}$.

1. **Garbling Phase:** Letting the garblers be $\mathcal{P}_{\mathsf{N}} = \{P_i\}_{i\in[n-1]}$, $\mathcal{P}$ invokes the garbling functionality $\mathcal{F}_{\mathsf{GS}}$ as in Figure 3 for $\mathsf{N} = n-1$.
   - Each $P_i \in \mathcal{P}_{\mathsf{N}}$ inputs to $\mathcal{F}_{\mathsf{GS}}$:
     - for each wire $w \in [n+q]$, values $\lambda^i_w, k^i_{w,0}, k^i_{w,1}$
     - $\forall j \in [\mathsf{N}], (a,b) \in \{0,1\}^2$, $g \in [q]$, $F^{i,j,g}_{a,b} = \mathsf{PRF}_{k^i_{A,a},k^i_{B,b}}(g||j)$
   - All $P_i \in \mathcal{P}$ get output from $\mathcal{F}_{\mathsf{GS}}$:
     - the garbled circuit $\{G^j_{a,b}\}_{j\in[\mathsf{N}],g\in[q],(a,b)\in\{0,1\}^2}$
     - colour bit for each output wire $\{\lambda_w\}_{w\in[n+q-m,n+q]}$
     - for input wire i, colour bit λ_i
2. **OT Phase:**
   - Each party $P_i \in \mathcal{P}_{\mathsf{N}}$ sends $\Lambda_i = x_i \oplus \lambda_i$ to P_n
   - P_n computes $\Lambda_n = x_n \oplus \lambda_n$
   - $\forall i \in [n], j \in [\mathsf{N}]$, $\mathcal{F}_{\mathsf{OT}}$ is invoked where P_n is the receiver with input Λ_i and P_j is the sender with inputs $(k^j_{i,0}, k^j_{i,1})$
   - After all $\mathcal{F}_{\mathsf{OT}}$ calls, P_n receives $\{k^j_{i,\Lambda_i}\}_{j\in[n-1],i\in[n]}$
3. **Evaluation Phase:** P_n evaluates the garbled circuit:
   - for each gate $g \in [q]$ in topological order, and public values a and b on the input wires A and B, for all $j \in [\mathsf{N}]$, compute
   $$c||k^j_{g,c} = G^j_{a,b} \oplus \left(\bigoplus_{i=1}^{n-1} \mathsf{PRF}_{k^i_{A,a},k^i_{B,b}}(g||j) \right)$$
   - for each output wire w with value Λ_w, compute actual value $\Lambda_w \oplus \lambda_w$
   - P_n sends $f(\vec{x})$ to all parties

Fig. 4. Semi-honest Fall-Back Secure Protocol

security of garbling holds only under computational assumptions, the adversary can derive (brute-force) all the randomness of this honest garbler, including the masking bit λ_i. In such a case, the adversary can easily derive the input of this honest party, violating privacy.

Lemma 3.4 *Let $\mathcal{Z}_{\mathsf{S}}$ be the set of all subsets of $\{P_i\}_{i\in[n-1]}$.*

- *Let Π_{GS} be an n-party protocol computing $\mathcal{F}_{\mathsf{GS}}$ that is secure against a semi-honest PPT adversary corrupting any subset of $n-1$ parties, with $\mathcal{P}_{\mathsf{N}} = \{P_i\}_{i\in[n-1]}$ as the set of garblers.*
- *Let Π_{OT} be a protocol computing $\mathcal{F}_{\mathsf{OT}}$ that is secure against a semi-honest unbounded sender and a semi-honest PPT receiver.*

Then Figure 4 securely computes any function f with fall-back security (Definition 3.1) tolerating $\mathcal{Z}_{\mathsf{S}}$ for semi-honest adversaries in the plain model when the functionalities $\mathcal{F}_{\mathsf{OT}}$ and $\mathcal{F}_{\mathsf{GS}}$ are replaced by Π_{OT} and Π_{GS} respectively.

The complete proof for this can be found in Appendix A.1.

Proof Outline. The protocol in Figure 4 is an n-party protocol in which the first $n-1$ parties engage in a multi-party garbling protocol that is secure in the presence of a PPT adversary corrupting any subset of these garblers. This sub-protocol outputs a garbling that is given to the last party. Each party additionally gives this last party a bit representing its input (but masks it). The last party uses this set of bits to engage in n instances of oblivious transfer with all other parties as senders to get the input labels for the garbling. Finally, this party evaluates the garbled circuit and gives everyone the output.

Firstly, note that this protocol is secure in the presence of a PPT semi-honest adversary corrupting any subset of the parties. This is derived from the fact that both the multi-party garbling sub-protocol and all instances of the OT protocols are secure in the presence of a PPT adversary corrupting any of the parties. A simulator, for this case, works by making black-box calls to the simulators of these sub-protocols.

It remains to handle the case of corruption by an unbounded adversary. We need to show that the protocol in Figure 4 is secure in the presence of an unbounded semi-honest adversary corrupting any subset of the garblers. To this effect, note that in all the OT protocol executions, the garblers only act as senders. The OT protocol is secure in the presence of an unbounded adversary corrupting the sender. The receiver is always the last (evaluating) party and this is never corrupted by the adversary. Next, note that the multi-party garbling sub-protocol only involves each garbler's randomness and is independent of the function input.

So a simulator in the presence of an unbounded adversary would work by first participating in the garbling protocol on behalf of the honest garblers, exactly as in the real protocol execution. Next, it would make black-box calls to the OT simulator for each execution of OT involving a corrupt garbler. Finally, it would send the output $f(x)$ to all corrupt parties.

4 Compiling to Semi-Honest Fall-Back Security

In this section, we will describe an MPC protocol for general functionalities that satisfies semi-honest fall-back security for $\mathcal{Z}_{\mathsf{S}}$ containing any subset of less than $\frac{n}{2}$ parties. At the heart of this protocol is a compiler that we construct that can upgrade any MPC protocol that is secure against semi-honest unbounded adversaries with an adversary structure $\mathcal{Z}_{\mathsf{S}}$ to a protocol with semi-honest fall-back security for $\mathcal{Z}_{\mathsf{S}}$. The final protocol combines two components: a 'virtual protocol' that is secure in the presence of semi-honest unbounded adversaries; and a 'real protocol' compiling this into a protocol with fall-back security.

Virtual Protocol. Let $\mathcal{V} = \{V_i\}_{i\in[n]}$ be a set of n virtual parties. Let Π_{stat} be an n-party protocol executed among $\mathcal{V}$ with security against a semi-honest unbounded adversary corrupting $< \frac{n}{2}$ of the parties in $\mathcal{V}$. For any function f

there exists such a protocol [BGW88] that computes it. For a party $V_i \in \mathcal{V}$, we refer to its complete view in Π_{stat} as $\mathbf{View}_{\Pi_{\mathsf{stat}},i}$ and by $\mathbf{View}^{j}_{\Pi_{\mathsf{stat}},i}$, we refer to the partial view of V_i in the protocol up to round j. This includes the security parameter κ, the input x_i, contents of the random tape r_i, and all the messages received from all parties in $\mathcal{V}$ up to round j.

$$\mathbf{View}^{j}_{\Pi_{\mathsf{stat}},i} = (\kappa, x_i, r_i, \{m^{j'}_{k\to i}\}_{j'\in[j],k\neq i\in[n]})$$

Here $m^{j'}_{k\to i}$ is the message that party V_k sends V_i in round j'. Let NxtMsg be the next message function such that,

$$\mathsf{NxtMsg}^{j}_{i}(\mathbf{View}^{j-1}_{\Pi_{\mathsf{stat}},i}) = \{m^{j}_{i\to k}\}_{k\neq i\in[n]}$$

This function is parameterized by $i \in [n]$ that indexes the virtual party $V_i \in \mathcal{V}$. It takes as input the view up to round $j-1$ and creates all the messages to be delivered to all the parties in round j. Note that after each round,

$$\mathbf{View}^{j}_{\Pi_{\mathsf{stat}},i} = \mathbf{View}^{j-1}_{\Pi_{\mathsf{stat}},i} \cup \{m^{j}_{k\to i}\}_{k\neq i\in[n]}$$

Letting r be the number of rounds in Π_{stat}, the complete protocol Π_{stat} itself can be described as a set of the next message functions,

$$\Pi_{\mathsf{stat}} = \{\mathsf{NxtMsg}^{j}_{i}(\mathbf{View}^{j-1}_{\Pi_{\mathsf{stat}},i})\}_{i\in[n],j\in[r]}$$

Finally, for any subset of parties $Z \subseteq \mathcal{V}$ we denote by $\mathbf{View}_{\Pi_{\mathsf{stat}},Z}$ the combined view of Π_{stat} for all the parties in Z and similarly, we denote by $\mathbf{View}^{j}_{\Pi_{\mathsf{stat}},Z}$ the combined view up to round j in the protocol.

Real Protocol. Let $\mathcal{P} = \{P_i\}_{i\in[n]}$ be n parties that need to compute a multi-party function f on $\vec{x} = \{x_i\}_{i\in[n]}$. Each party $P_i \in \mathcal{P}$ has input x_i. Let Π_{stat} be an n-party protocol for the function f defined as follows:

- Each party $V_i \in \mathcal{V}$ has the input bit x_i and this is given to $\mathcal{F}$.
- $\mathcal{F}$ uses $\vec{x} = \{x_i\}_{i\in[n]}$ to compute $f(\vec{x})$ and gives it to each party $V_i \in \mathcal{V}$.

This protocol Π_{stat} can be re-written as a set of next message functions $\Pi_{\mathsf{stat}} = \{\mathsf{NxtMsg}^{j}_{i}\}_{i\in[n],j\in[r]}$. Let $\mathsf{SS}_{n,n} = (\mathsf{Share}_{n,n}, \mathsf{Recon}_{n,n})$ be an n-out-of-n secret-sharing scheme. For each $i \in [n], j \in [r]$ let $\mathcal{F}_{\mathsf{NxtMsg}^{j}_{i}}$ be a functionality as in Figure 5.

Note that $\mathcal{F}_{\mathsf{NxtMsg}^{j}_{i}}$ is a functionality whose output is symmetric. Let Π be a protocol that computes any functionality exactly as in Figure 4. For each $i \in [n]$, let Π_i denote the protocol Π with party P_i playing the role of the evaluator and all other parties as the garbler. Let $\Pi_{\mathsf{NxtMsg}^{j}_{i}}$ denote a protocol Π_i executed to compute the functionality $\mathcal{F}_{\mathsf{NxtMsg}^{j}_{i}}$. That is, it is an n-party protocol with $n-1$ garblers and one evaluator that is secure against the corruption of any subset of the garblers by a semi-honest unbounded adversary, and the corruption of any subset of parties by a PPT semi-honest adversary. The protocol $\Pi_{\mathsf{NxtMsg}^{j}_{i}}$

Next Message Functionality $\mathcal{F}_{\mathsf{NxtMsg}_i^j}$

Setting: Let $\mathcal{P} = \{P_i\}_{i\in[n]}$ be n parties. Let $\mathsf{SS}_{n,n} = (\mathsf{Share}_{n,n}, \mathsf{Recon}_{n,n})$ be an n-out-of-n secret-sharing scheme. For a protocol Π_{stat} computing $\mathcal{F}$ let

$$\mathsf{NxtMsg}_i^j(\mathbf{View}^{j-1}_{\Pi_{\mathsf{stat}},i}) = \{m^j_{i\to k}\}_{k\neq i\in[n]}$$

be the next message function for party V_i in round j. The n-party functionality $\mathcal{F}_{\mathsf{NxtMsg}_i^j}$ operates as follows:

- **Inputs:** Each party $P_k \in \mathcal{P}$ has:
  - Let $\ell' = |\{[m^j_{i\to i'}]_k\}_{i'\neq i\in[n]}|$. Sample a random mask $M_k \leftarrow \{0,1\}^{\ell'}$.
  - Share of the view of V_i until round $(j-1)$ in Π_{stat},

$$[\mathbf{View}^{j-1}_{\Pi_{\mathsf{stat}},i}]_k = \left(\kappa, [x_i]_k, [r_i]_k, \{[m^{j'}_{i'\to i}]_k\}_{j'\in[j-1],i'\neq i\in[n]}\right)$$

  P_k gives to $\mathcal{F}_{\mathsf{NxtMsg}_i^j}$ the inputs,

$$[\mathbf{View}^{j-1}_{\Pi_{\mathsf{stat}},i}]_k, M_k$$

- **Computation:** The functionality $\mathcal{F}_{\mathsf{NxtMsg}_i^j}$ computes:

$$\begin{aligned} \mathbf{View}^{j-1}_{\Pi_{\mathsf{stat}},i} &\leftarrow \mathsf{Recon}_{n,n}(\{[\mathbf{View}^{j-1}_{\Pi_{\mathsf{stat}},i}]_k\}_{k\in[n]}) \\ \{m^j_{i\to i'}\}_{i'\neq i\in[n]} &= \mathsf{NxtMsg}_i^j(\mathbf{View}^{j-1}_{\Pi_{\mathsf{stat}},i}) \\ \forall i'\neq i\in[n], \quad \{[m^j_{i\to i'}]_k\}_{k\in[n]} &\leftarrow \mathsf{Share}_{n,n}(m^j_{i\to i'}) \\ \vec{c} &= \{c_k = M_k \oplus \{[m^j_{i\to i'}]_k\}_{i'\neq i\in[n]}\}_{k\in[n]} \end{aligned}$$

- **Output:** $\mathcal{F}_{\mathsf{NxtMsg}_i^j}$ gives $\vec{c}$ to each $P_k \in \mathcal{P}$.

Fig. 5. Next Message Functionality

is executed by the parties $\mathcal{P} = \{P_k\}_{k\in[n]}$ and the party P_i is designated as the evaluator, while all other parties are the garblers.

Combined Protocol. The complete protocol Π_{in} is the set of all the protocols $\{\Pi_{\mathsf{NxtMsg}_i^j}\}_{i\in[n],j\in[r]}$ for any virtual protocol Π_{stat} with r rounds computing $\mathcal{F}$. It is formally described in Figure 6.

Theorem 4.1 *Let Π_{stat} be a n-party protocol that securely computes a function f in the presence of an unbounded semi-honest adversary with adversary structure $\mathcal{Z}_{\mathsf{S}}$. Assuming protocol Π is an n-party protocol that computes any function with fall-back security with respect to $\mathcal{Z}'_{\mathsf{S}} = \{Z \subseteq \{P_i\}_{i\in[n-1]}\}$, the protocol in Figure 6 securely compiles Π_{stat} to compute f with fall-back security (Definition 3.1) tolerating $\mathcal{Z}_{\mathsf{S}}$ in the presence of semi-honest adversaries.*

A detailed proof for this can be found in Appendix A.2.

Compiler for Semi-honest Fall-Back Security

Setting: For function f on $\vec{x} = \{x_i\}_{i\in[n]}$, let $\Pi_{\mathsf{stat}} = \{\mathsf{NxtMsg}_i^j\}_{i\in[n],j\in[r]}$ be a protocol computing the functionality $\mathcal{F}$ among $\mathcal{V} = \{V_i\}_{i\in[n]}$. Π_{stat} securely computes $\mathcal{F}$ against a semi-honest unbounded adversary corrupting $< \frac{n}{2}$ parties. Let $\mathcal{P} = \{P_i\}_{i\in[n]}$ be the set of real parties where P_i has input x_i.

1. **Initialize shared state.** Each party $P_i \in \mathcal{P}$ does the following:
   – Compute $\mathsf{Share}_{n,n}(x_i) \rightarrow \{[x_i]_{i'}\}_{i'\in[n]}$
   – Sample $r_i \leftarrow \mathcal{R}(\kappa)$ and compute $\mathsf{Share}_{n,n}(r_i) \rightarrow \{[r_i]_{i'}\}_{i'\in[n]}$
   – $\forall i' \neq i \in [n]$, send $([x_i]_{i'}, [r_i]_{i'})$ to $P_{i'}$
   – $\forall k \in [n]$, set the share of the initial state of virtual party $V_k \in \mathcal{V}$ as,
   $$[\mathbf{View}^0_{\Pi_{\mathsf{stat}},k}]_i = ([x_k]_i, [r_k]_i)$$
2. **Compute virtual protocol.** For each $\mathsf{NxtMsg}_i^j \in \Pi_{\mathsf{stat}}$, let $\Pi_{\mathsf{NxtMsg}_i^j}$ be an n-party protocol as in Figure 4 with P_i as the evaluator and $\mathcal{P}_{\mathsf{N}} = \mathcal{P} - \{P_i\}$ as the garblers. Each party $P_{i'} \in \mathcal{P}$ does the following:
   – $P_{i'}$ has input $[\mathbf{View}^{j-1}_{\Pi_{\mathsf{stat}},i}]_{i'}$ and samples $M_{i'} \leftarrow \{0,1\}^{\ell'}$
   – $P_{i'}$ participates in an execution of $\Pi_{\mathsf{NxtMsg}_i^j}$ with input
   $$[\mathbf{View}^{j-1}_{\Pi_{\mathsf{stat}},i}]_{i'}, M_{i'}$$
   Let $\mathbf{View}_{\mathsf{NxtMsg}_i^j,i'}$ be the view of $P_{i'}$ in this protocol.
   – $P_{i'}$ gets the output $\vec{c} = \{c_k = M_k \oplus \{[m^j_{i\rightarrow k'}]_k\}_{k'\neq i\in[n]}\}_{k\in[n]}$
   – $P_{i'}$ extracts $\{[m^j_{i\rightarrow k'}]_{i'}\}_{k'\neq i\in[n]} = c_{i'} \oplus M_{i'}$
   – $P_{i'}$ updates the view for each virtual party $V_k \in \mathcal{V} - \{V_i\}$,
   $$[\mathbf{View}^j_{\Pi_{\mathsf{stat}},k}]_{i'} = [\mathbf{View}^j_{\Pi_{\mathsf{stat}},k}]_{i'} \cup \{[\mathbf{View}^{j-1}_{\Pi_{\mathsf{stat}},k}]_{i'}, [m^j_{i\rightarrow k}]_{i'}\}$$
   $\Pi_{\mathsf{NxtMsg}_i^j}$ is computed for all $V_i \in \mathcal{V}$ and rounds indexed $j \in [r]$ in Π_{stat}
3. **Derive the output.** After the complete virtual protocol is computed, each party $P_i \in \mathcal{P}$ has a share of the view of each virtual party,
   $$\{[\mathbf{View}_{\Pi_{\mathsf{stat}},j}]_i\}_{j\in[n]}$$
   – Each party $P_i \in \mathcal{P}$ extracts the shares of the output:
   $$\{[f_j(\vec{x})]_i\}_{j\in[n]} \leftarrow \{[\mathbf{View}_{\Pi_{\mathsf{stat}},j}]_i\}_{j\in[n]}$$
   – P_i sends $[f_j(\vec{x})]_i$ to $P_j \in \mathcal{P}$
   – On receiving $\{[f_i(\vec{x})]_j\}_{j\in[n]}$, P_i reconstructs the output $f_i(\vec{x}) = f(\vec{x})$

Fig. 6. Compiler for Semi-honest Fall-Back Security

Proof Outline. Firstly, the compiled protocol as in Figure 6 is a sequence of protocols Π that are composed using n-out-of-n secret sharing of the views of the parties participating in the virtual protocol Π_{stat}. Each such protocol Π

satisfies fall-back security in the presence of semi-honest adversaries. This, along with the fact that an adversary not corrupting all the parties does not learn any information about a secret from its shares, guarantees that the compiled protocol, Figure 6, is secure in the presence of a semi-honest PPT adversary corrupting any subset of the parties in $\mathcal{P}$.

It remains to handle the case of corruption by an unbounded adversary. The protocol Π tolerates an adversary structure of the form $\mathcal{Z}'_{\mathsf{S}}$ that includes all subsets of $\mathcal{P}$ that exclude a designated party P_i. Intuitively, in Figure 6, if a semi-honest unbounded adversary $\mathcal{A}$ corrupts any subset of parties $Z \in \mathcal{Z}_{\mathsf{S}}$ where $Z \subset \mathcal{P}$, this corresponds to corrupting every execution of Π in which a corrupted party $P_i \in Z$ is the party excluded in the adversary structure $\mathcal{Z}'_{\mathsf{S}}$. That is, the adversary has in its view all the inputs, randomness, and messages for all these executions. Corrupting each such P_i in Π_{in} corresponds to corrupting V_i in Π_{stat} since the corrupt executions of Π completely reveal the view of V_i.

For every $P_i \notin Z$, no information about view of V_i is revealed. This is due to the fact that the views of all virtual parties are shared using n-out-of-n secret sharing and the unbounded adversary, not having the honest parties' shares, has no information about the view. Furthermore, for every execution of Π corresponding to such a NxtMsg_i^j, the set of corrupt parties $Z \in \mathcal{P}$ are such that $Z \in \mathcal{Z}'_{\mathsf{S}}$. Finally, note that the view of parties $Z \in \mathcal{Z}_{\mathsf{S}}$ where $Z \subset \mathcal{V}$ is simulatable since Π_{stat} tolerates $\mathcal{Z}_{\mathsf{S}}$. So it follows that the view of the compiled protocol is also simulatable.

For any protocol Π_{stat} for any set of parties $Z \in \mathcal{Z}_{\mathsf{S}}$, there would exist a PPT simulator $\mathsf{Sim}_{\mathsf{stat}}$ that can simulate the view of the protocol in a way that is statistically close to the real view. This simulator can also be written in terms of next-message functions. The protocol Π computes any n-party functionality and so can compute these simulated next-message functions as well. A simulator for the compiled protocol, Figure 6, in the presence of an unbounded semi-honest protocol, would make black-box calls to simulators for the view of Π executing the simulated next-message functions of the virtual protocol. This can be shown as statistically close to the view in the real execution of the protocol using a sequence of hybrids first replacing each simulated view of Π with the real view. Finally, the set of next-message functions of the virtual protocol simulation are replaced by the next-message functions of an actual virtual protocol execution.

Round Complexity. In Figure 6, let r be the number of rounds that the virtual protocol Π_{stat} takes. Let c be the number of rounds of protocol Π (Figure 4). Then Figure 6 requires $cr+2$ rounds. Our result implies that assuming a constant-round statistical oblivious-transfer, then protocol Π (Figure 4) is constant round and our compiler (Figure 6) yields a protocol with linear round complexity in the number of rounds in the virtual (information-theoretic) protocol. In comparison, previous work [CDvdG87] solves the former with a number of rounds proportional to the number of participating parties. Using their protocol within our compiler would yield a number of rounds proportional to both the number of parties and rounds in the virtual protocol. We remark here that we essentially achieve the best possible round complexity (up to constant factors).

Corollary 4.2 *Let Π_{stat} be a n-party protocol that securely computes a function f in the presence of an unbounded semi-honest adversary with adversary structure $\mathcal{Z}_{\mathsf{S}}$. For each $i \in [n]$, let Π_i be an n-party protocol that can compute any functionality with semi-honest fall-back security with respect to the adversary structure $\mathcal{Z}_{\mathsf{S}}^i = \{Z : Z \in \mathcal{Z}_{\mathsf{S}}, P_i \notin Z\}$. Then Figure 6 securely compiles Π_{stat} to compute f with fall-back security (Definition 3.1) with respect to $\mathcal{Z}_{\mathsf{S}}$ and in the presence of semi-honest adversaries.*

5 MPC with Fall-Back Security – Malicious Security

This section will describe an MPC protocol for general functionalities that satisfies malicious fall-back security. Let $\mathcal{Z}_{\mathsf{S}}$ be any adversary structure for which there exists a protocol for computing any functionality in the presence of a malicious unbounded adversary tolerating $\mathcal{Z}_{\mathsf{S}}$ [HM97]. We construct a protocol that is maliciously fall-back secure with respect to $\mathcal{Z}_{\mathsf{S}}$.

Our protocol works in the offline-online paradigm with a function-and-input-independent pre-processing phase based on [HVW20]. This phase has an n-party protocol computing the authenticated triples functionality $\mathcal{F}_{\mathsf{AuthTriples}}$ as given in Figure 7 with malicious fall-back security tolerating $\mathcal{Z}_{\mathsf{S}}$. Then, given the output of this phase, we use a protocol in the online phase, based on that in [DPSZ12], that computes any function f with malicious fall-back security. Our focus here is on designing the protocol for the pre-processing phase and providing a construction for the commitment scheme used in the online phase.

Pre-Processing. This phase contains an n-party protocol for computing authenticated multiplication triples with malicious fall-back security tolerating $\mathcal{Z}_{\mathsf{S}}$. The protocol conceptually contains three main components:

- **Authenticated Triples Generation Functionality.** Our starting point is the Authenticated Triples functionality $\mathcal{F}_{\mathsf{AuthTriples}}$ as given in Figure 7. This is the n-party functionality that the pre-processing phase aims to realize. Details can be found in Section 5.1.
- **Semi-honest adaptively secure protocol with fall-back security.** We require a protocol realizing $\mathcal{F}_{\mathsf{AuthTriples}}$ that is fall-back secure tolerating $\mathcal{Z}_{\mathsf{S}}$ in the presence of semi-honest adaptive adversaries. Our starting point for this is an n-client-m-server protocol realizing $\mathcal{F}_{\mathsf{AuthTriples}}$ in the presence of an unbounded semi-honest adversary that can corrupt any number of the clients and up to a threshold t of the servers (Protocol 5.1). In this protocol, each server works by receiving inputs from all the clients, performing local computation, and giving outputs to these clients. As such, the actions of each server can be abstracted into a functionality $\mathcal{F}_S$ (Figure 8). The final semi-honest fall-back secure protocol for $\mathcal{F}_{\mathsf{AuthTriples}}$ that we need works by having all the n parties act as the clients and also jointly compute $\mathcal{F}_S$ for each of the m servers. This computation of $\mathcal{F}_S$ is done by a semi-honest fall-back secure protocol that is secure in the presence of adaptive adversaries. This is discussed in detail in Section 5.2.

– **Compiling to maliciously secure fall-back security.** The adaptively secure protocol with semi-honest fall-back security as described above is then lifted to provide malicious fall-back security tolerating the same adversary structure $\mathcal{Z}_{\mathsf{S}}$. This is done using an n-party commitment protocol with fall-back security tolerating $\mathcal{Z}_{\mathsf{S}}$ as a building block. The construction for this is discussed in Section 5.3. The details of the final pre-processing protocol can be found in Section 5.4.

Online Phase. Given the output of the pre-processing phase, our online protocol (Figure 13) follows the online phase closely in [DPSZ12] that can compute any functionality in the presence of an unbounded malicious adversary corrupting any subset of the parties in the $\mathcal{F}_{\mathsf{Com}}$-hybrid, with $\mathcal{F}_{\mathsf{Com}}$ being an n-party commitment functionality. We describe our protocol in the plain model, instantiating $\mathcal{F}_{\mathsf{Com}}$ with a commitment protocol with fall-back security tolerating $\mathcal{Z}_{\mathsf{S}}$.

5.1 Authenticated Triples Generation

We describe an n-party Authenticated Triples Generation functionality $\mathcal{F}_{\mathsf{AuthTriples}}$ (Figure 7). Let f be a function and $\vec{x}$ be its input. Let $\mathcal{P} = \{P_i\}_{i\in[n]}$ where each party P_i has private input $x_i \in \vec{x}$ such that $x_i \in \{0,1\}$. Let $\mathbf{C}$ be a circuit representing f using only binary addition and multiplication gates. Let $\mathsf{T} \in \mathbb{N}$ be the number of multiplication gates in $\mathbf{C}$. The functionality $\mathcal{F}_{\mathsf{AuthTriples}}$ creates an n-out-of-n additive sharing of T random multiplication triples of the form $\{a^j, b^j, c^j = (a^j \cdot b^j)\}_{j\in[\mathsf{T}]}$ and gives one share to each party. Additionally, it additively secret-shares a MAC key Δ and MACs on the multiplication triples $\{\mathsf{MAC}(a^j) = (a^j \cdot \Delta),\ \mathsf{MAC}(b^j) = (b^j \cdot \Delta),\ \mathsf{MAC}(c^j) = (c^j \cdot \Delta)\}_{j\in[\mathsf{T}]}$ among the parties. Finally, for each input bit x_i, it samples a random number r_i and gives it to P_i. It distributes among all parties a sharing of r_i and $\mathsf{MAC}(r^i) = (r^i \cdot \Delta)$. The functionality first allows corrupted parties to sample their own shares and then samples shares for the honest parties, setting the shares of a designated party P_1 such that the constraints are met.

5.2 Authenticated Triples with Semi-Honest Fall-Back Security

In this section, we describe the protocol computing $\mathcal{F}_{\mathsf{AuthTriples}}$ with semi-honest fall-back security tolerating $\mathcal{Z}_{\mathsf{S}}$ for adaptive corruption. This is the starting point from which we lift semi-honest to malicious security. However, before discussing the fall-back secure protocol, consider the semi-honest protocol as in Protocol 5.1.

For a security parameter κ, let $t = \kappa$ be a threshold and set $m = 16t$. Protocol 5.1 is a semi-honest secure protocol for computing $\mathcal{F}_{\mathsf{AuthTriples}}$ among n clients $\mathcal{P} = \{P_i\}_{i\in[n]}$ and m virtual servers $\mathcal{S} = \{S_j\}_{j\in[m]}$. In this protocol, the clients have the inputs and receive the outputs of the functionality and the servers only aid in computation. The protocol works by having each client sample its inputs and use t-out-of-m threshold secret-sharing to share it among the servers.

Authenticated Triples Generation $\mathcal{F}_{\mathsf{AuthTriples}}$

Setting: Let $\mathcal{P} = \{P_i\}_{i\in[n]}$ be a set of parties. Let f be a function and $\vec{x} \in \{0,1\}^n$ be its input such that each party $P_i \in \mathcal{P}$ has input $x_i \in \vec{x}$. Let $\mathbb{F}$ be a field and $\mathsf{T} \in \mathbb{N}$ be the number of multiplication gates in a circuit $\mathbf{C}$ representing f. Let $Z \subset \mathcal{P}$ be the corrupt parties. The functionality $\mathcal{F}_{\mathsf{AuthTriples}}$ operates as follows:

- **Inputs:** Each corrupt party $P_i \in Z$ samples:
  - $\forall j \in [\mathsf{T}]$, triples shares $a_i^j, b_i^j, c_i^j, \mathsf{MAC}_i(a^j), \mathsf{MAC}_i(b^j), \mathsf{MAC}_i(c^j) \in \mathbb{F}$
  - $\forall k \in [n]$, input mask shares $r_i^k, \mathsf{MAC}_i(r^k) \in \mathbb{F}$
  - key share $\Delta_i \in \mathbb{F}$
- **Computation:** The functionality $\mathcal{F}_{\mathsf{AuthTriples}}$ computes,
  - $\forall P_i \in \mathcal{P} - Z$, sample

$$\Delta_i \leftarrow \mathbb{F}, \{r_i^k, \mathsf{MAC}_i(r^k) \leftarrow \mathbb{F}\}_{k\in[n]}$$
$$\{a_i^j, b_i^j, c_i^j, \mathsf{MAC}_i(a^j), \mathsf{MAC}_i(b^j), \mathsf{MAC}_i(c^j) \leftarrow \mathbb{F}\}_{j\in[\mathsf{T}]}$$

  - Compute,

$$\begin{aligned}
\forall k \in [n], \qquad r^k &= \sum_{i\in[n]} r_i^k \\
\mathsf{MAC}'(r^k) &= (\sum_{i\in[n]} r_i^k) \cdot (\sum_{i\in[n]} \Delta_i) - (\sum_{i\in[n]} \mathsf{MAC}_i(r^k)) \\
\forall j \in [\mathsf{T}], \qquad c'^j &= (\sum_{i\in[n]} a_i^j) \cdot (\sum_{i\in[n]} b_i^j) - (\sum_{i\in[n]} c_i^j) \\
\mathsf{MAC}'(a^j) &= (\sum_{i\in[n]} a_i^j) \cdot (\sum_{i\in[n]} \Delta_i) - (\sum_{i\in[n]} \mathsf{MAC}_i(a^j)) \\
\mathsf{MAC}'(b^j) &= (\sum_{i\in[n]} b_i^j) \cdot (\sum_{i\in[n]} \Delta_i) - (\sum_{i\in[n]} \mathsf{MAC}_i(b^j)) \\
\mathsf{MAC}'(c^j) &= (\sum_{i\in[n]} a_i^j) \cdot (\sum_{i\in[n]} b_i^j) \cdot (\sum_{i\in[n]} \Delta_i) - (\sum_{i\in[n]} \mathsf{MAC}_i(c^j))
\end{aligned}$$

- **Outputs:** $\mathcal{F}_{\mathsf{AuthTriples}}$ gives to each $P_i \neq P_1 \in \mathcal{P}$,

$$\Delta_i, r^i, \{r_i^k, \mathsf{MAC}_i(r^k)\}_{k\in[n]}, \{a_i^j, b_i^j, c_i^j, \mathsf{MAC}_i(a^j), \mathsf{MAC}_i(b^j), \mathsf{MAC}_i(c^j)\}_{j\in[\mathsf{T}]}$$

  $\mathcal{F}_{\mathsf{AuthTriples}}$ gives to P_1,

$$\Delta_1, r^1, \{r_1^k, \mathsf{MAC}_1(r^k) = \mathsf{MAC}_1(r^k) + \mathsf{MAC}'(r^k)\}_{k\in[n]},$$
$$\{a_1^j, b_1^j, c_1^j = c_1^j + c'^j, \mathsf{MAC}_1(a^j) = \mathsf{MAC}_1(a^j) + \mathsf{MAC}'(a^j),$$
$$\mathsf{MAC}_1(b^j) = \mathsf{MAC}_1(b^j) + \mathsf{MAC}'(b^j), \mathsf{MAC}_1(c^j) = \mathsf{MAC}_1(c^j) + \mathsf{MAC}'(c^j)\}_{j\in[\mathsf{T}]}$$

Fig. 7. Authenticated Triples Generation

Then each server performs some symmetric local computation to produce shares of the outputs of the functionality. The actions of each server can be captured by an n-party functionality $\mathcal{F}_{\mathcal{S}}$ (Figure 8). Each server then gives these shares to the respective clients and each client reconstructs its outputs.

Protocol 5.1 *For a security parameter κ, let $t = \kappa$ be a threshold and for $m > 16t$, let $\mathcal{S} = \{S_i\}_{i\in[m]}$ be a set of m virtual servers. Let $\mathsf{SS}_{t,m} = (\mathsf{Share}_{t,m}, \mathsf{Recon}_{t,m})$ be a t-out-of-m threshold secret-sharing scheme [Sha79]. Let $\mathcal{V} = \{V_i\}_{i\in[n]}$ be a set of n virtual parties.*

– *Each party $V_i \in \mathcal{V}$ first samples,*

$$\Delta_i \leftarrow \mathbb{F}, \left\{r_i^k, \mathsf{MAC}_i(r^k) \leftarrow \mathbb{F}\right\}_{k\in[n]}$$
$$\left\{a_i^j, b_i^j, c_i^j, \mathsf{MAC}_i(a^j), \mathsf{MAC}_i(b^j), \mathsf{MAC}_i(c^j) \leftarrow \mathbb{F}\right\}_{j\in[\mathsf{T}]}$$

It then computes a $3t$-out-of-m threshold secret-sharing of each element in $\left\{\mathsf{MAC}_i(c^j)\right\}_{j\in[\mathsf{T}]}$ and a t-out-of-m threshold secret-sharing of all other elements.

– *Each party $V_i \in \mathcal{V}$ gives to server $S_j \in \mathcal{S}$ the shares,*

$$[\Delta_i]_j, \left\{[r_i^k]_j, [\mathsf{MAC}_i(r^k)]_j\right\}_{k\in[n]}$$
$$\left\{[a_i^{j'}]_j, [b_i^{j'}]_j, [c_i^{j'}]_j, [\mathsf{MAC}_i(a^{j'})]_j, [\mathsf{MAC}_i(b^{j'})]_j, [\mathsf{MAC}_i(c^{j'})]_j\right\}_{j'\in[\mathsf{T}]}$$

– *Each server $S_j \in \mathcal{S}$ computes the functionality $\mathcal{F}_{\mathcal{S}}$ (Figure 8). It gives to party $V_1 \in \mathcal{V}$ the values,*

$$[r^1]_j, \{[\mathsf{MAC}'(r^k)]_j\}_{k\in[n]}, \left\{[c'^{j'}]_j, [\mathsf{MAC}'(a^{j'})]_j, [\mathsf{MAC}'(b^{j'})]_j, [\mathsf{MAC}'(c^{j'})]_j\right\}_{j'\in[\mathsf{T}]}$$

It gives each party $V_i \in \mathcal{V}$ the share $[r^i]_j$.

– *The party V_1 reconstructs,*

$$\Delta_1, r^1, \{r_1^k, \mathsf{MAC}_1(r^k) = \mathsf{MAC}_1(r^k) + \mathsf{MAC}'(r^k)\}_{k\in[n]},$$
$$\left\{a_1^j, b_1^j, c_1^j = c_1^j + c'^j, \mathsf{MAC}_1(a^j) = \mathsf{MAC}_1(a^j) + \mathsf{MAC}'(a^j),\right.$$
$$\left.\mathsf{MAC}_1(b^j) = \mathsf{MAC}_1(b^j) + \mathsf{MAC}'(b^j), \mathsf{MAC}_1(c^j) = \mathsf{MAC}_1(c^j) + \mathsf{MAC}'(c^j)\right\}_{j\in[\mathsf{T}]}$$

– *Each $V_i \neq V_1$ reconstructs r^i.*

Lemma 5.2 *Protocol 5.1 securely realizes functionality $\mathcal{F}_{\mathsf{AuthTriples}}$ (Figure 7) in the presence of an unbounded semi-honest adversary corrupting all-but-one parties in $\mathcal{V}$ and any subset of t servers in $\mathcal{S}$.*

The semi-honest secure Protocol 5.1 is secure in the presence of a semi-honest unbounded adversary corrupting all-but-one of the clients and up to t servers. We define a protocol Π_S to be an n-party protocol realizing $\mathcal{F}_{\mathcal{S}}$ (Figure 8) with semi-honest fall-back security tolerating $\mathcal{Z}_{\mathsf{S}}$ in the presence of adaptive corruptions. Given such a protocol, we can define a protocol $\Pi_{\mathsf{AuthTriples}}$ that realizes $\mathcal{F}_{\mathsf{AuthTriples}}$ with semi-honest fall-back security tolerating $\mathcal{Z}_{\mathsf{S}}$ for adaptive adversaries. This protocol would work by first having each of the n parties generate their inputs and create t-out-of-m threshold secret-shares. Then execute m instances of Π_S, each with a different set of shares as input. The output of these executions is reconstructed to get the output of $\mathcal{F}_{\mathsf{AuthTriples}}$.

Virtual Server Functionality $\mathcal{F}_S$

Setting: Let κ be a computational security parameter, $\mathsf{T} \in \mathbb{N}$ and $\mathbb{F}$ be a field. Let $\mathcal{V} = \{V_i\}_{i\in[n]}$ be a set of parties. The functionality $\mathcal{F}_S$ operates as follows:

– **Inputs:** Each party $V_i \in \mathcal{V}$ has:

$$[\Delta_i]_j, \big\{[r_i^k]_j, [\mathsf{MAC}_i(r^k)]_j\big\}_{k\in[n]}$$

$$\big\{[a_i^{j'}]_j, [b_i^{j'}]_j, [c_i^{j'}]_j, [\mathsf{MAC}_i(a^{j'})]_j, [\mathsf{MAC}_i(b^{j'})]_j, [\mathsf{MAC}_i(c^{j'})]_j\big\}_{j'\in[\mathsf{T}]}$$

as in Protocol 5.1.

– **Computation:** The functionality $\mathcal{F}_S$ computes,

$$\forall k \in [n], \qquad [r^k]_j = \sum_{i\in[n]} [r_i^k]_j$$

$$[\mathsf{MAC}'(r^k)]_j = \Big(\sum_{i\in[n]} [r_i^k]_j\Big) \cdot \Big(\sum_{i\in[n]} [\Delta_i]_j\Big) - \Big(\sum_{i\in[n]} [\mathsf{MAC}_i(r^k)]_j\Big)$$

$$\forall j' \in [\mathsf{T}], \qquad [c'^{j'}]_j = \Big(\sum_{i\in[n]} [a_i^{j'}]_j\Big) \cdot \Big(\sum_{i\in[n]} [b_i^{j'}]_j\Big) - \Big(\sum_{i\in[n]} [c_i^{j'}]_j\Big)$$

$$[\mathsf{MAC}'(a^{j'})]_j = \Big(\sum_{i\in[n]} [a_i^{j'}]_j\Big) \cdot \Big(\sum_{i\in[n]} [\Delta_i]_j\Big) - \Big(\sum_{i\in[n]} [\mathsf{MAC}_i(a^{j'})]_j\Big)$$

$$[\mathsf{MAC}'(b^{j'})]_j = \Big(\sum_{i\in[n]} [b_i^{j'}]_j\Big) \cdot \Big(\sum_{i\in[n]} [\Delta_i]_j\Big) - \Big(\sum_{i\in[n]} [\mathsf{MAC}_i(b^{j'})]_j\Big)$$

$$[\mathsf{MAC}'(c^{j'})]_j = \Big(\sum_{i\in[n]} [a_i^{j'}]_j\Big) \cdot \Big(\sum_{i\in[n]} [b_i^{j'}]_j\Big) \cdot \Big(\sum_{i\in[n]} [\Delta_i]_j\Big) - \Big(\sum_{i\in[n]} [\mathsf{MAC}_i(c^{j'})]_j\Big)$$

– **Outputs:** $\mathcal{F}_S$ gives to V_1:

$$[r^1]_j, \{[\mathsf{MAC}'(r^k)]_j\}_{k\in[n]}\{[c'^{j'}]_j, [\mathsf{MAC}'(a^{j'})]_j, [\mathsf{MAC}'(b^{j'})]_j, [\mathsf{MAC}'(c^{j'})]_j\}_{j'\in[\mathsf{T}]}$$

It gives $[r^i]_j$ to each $V_i \neq V_1 \in \mathcal{V}$.

Fig. 8. Virtual Server Functionality

Remark 5.3 (A note on adaptive security.) *Note that our final protocol requires this semi-honest secure protocol to satisfy adaptive security for the same reason the IPS compiler does. Recall that every virtual party is emulated via this semi-honest secure inner protocol, where the adversary can choose to deviate at any point. This implies that the simulator of this virtual protocol needs to provide the randomness for these corrupted parties consistent with the inputs provided by the real protocol simulator upon corruption. In Section 5.5, we show that such a simulation can be performed without actually requiring adaptive security for the individual primitives (garbled circuit and oblivious transfer) by carefully considering the different adaptive corruption scenarios.*

Commitment Functionality $\mathcal{F}_{\mathsf{Com}}$

Setting: Let $\mathcal{P} = \{P_i\}_{i \in [n]}$ be a set of parties. Let $C \in \mathcal{P}$ be the committer with input $x \in \mathcal{X}$. The functionality $\mathcal{F}_{\mathsf{Com}}$ operates as follows:

- **Inputs:** C has input x. All other parties have no input.
- **Commit Phase:** C gives x or $\perp$ to $\mathcal{F}_{\mathsf{Com}}$.
- **Decommit Phase:** $\mathcal{F}_{\mathsf{Com}}$ gives x to all parties in $\mathcal{P}$ if input is not $\perp$.

Fig. 9. n-party Commitment Functionality

Commitment Protocol with Fall-Back Security

Setting: Let $\mathcal{P} = \{P_i\}_{i \in [n]}$ and $\mathcal{Z}_{\mathsf{S}} \subset 2^{\mathcal{P}}$. Let $C \in \mathcal{P}$ be the committer with input $x \in \mathcal{X}$. There exist secure point-to-point and broadcast channels for all parties. Let $\mathsf{Com}_{\mathsf{SB}}$ (Definition 2.10) be a statistically binding commitment. For the adversary structure $\mathcal{Z}_{\mathsf{S}}$ let $\mathsf{SS}_{\mathcal{Z}_{\mathsf{S}}} = (\mathsf{Share}_{\mathcal{Z}_{\mathsf{S}}}, \mathsf{Recon}_{\mathcal{Z}_{\mathsf{S}}})$ be a secret-sharing scheme over $\mathcal{X}$. Let $\mathsf{Com}_{\mathsf{SH}}$ (Definition 2.11) be a statistically hiding commitment. The commitment protocol Π_{Com} operates as follows:

1. **Commit Phase.**
   - C computes shares $\{[x]_k\}_{k \in [n]} \leftarrow \mathsf{Share}_{\mathcal{Z}_{\mathsf{S}}}(x)$.
   - For each $P_k \in \mathcal{P}$ such that $P_k \neq C$, C samples randomness r^k_{Com} and computes $c^k_{\mathsf{Com}_{\mathsf{SB}}} = \mathsf{Com}_{\mathsf{SB}}([x]_k; r^k_{\mathsf{Com}})$.
   - C samples $r_{\mathsf{Com}_{\mathsf{SH}}}$ and broadcasts $c_{\mathsf{Com}_{\mathsf{SH}}} = \mathsf{Com}_{\mathsf{SH}}(x; r_{\mathsf{Com}_{\mathsf{SH}}})$.
2. **Decommit Phase.**
   - C sends r^k_{Com} to each $P_k \in \mathcal{P}$ and broadcasts $(r_{\mathsf{Com}_{\mathsf{SH}}}, x, \{[x]_k\}_{k \in [n]})$.
   - Each party $P_k \neq C \in \mathcal{P}$, checks if

$$\begin{aligned} x &== \mathsf{Recon}_{\mathcal{Z}_{\mathsf{S}}}(\{[x]_k\}_{k \in [n]}) \\ c_{\mathsf{Com}_{\mathsf{SH}}} &== \mathsf{Com}_{\mathsf{SH}}(x; r_{\mathsf{Com}_{\mathsf{SH}}}) \\ c^k_{\mathsf{Com}_{\mathsf{SB}}} &== \mathsf{Com}_{\mathsf{SB}}([x]_k; r^k_{\mathsf{Com}}) \end{aligned}$$

     If this holds, accept x as output.

Fig. 10. Commitment Protocol with Fall-Back Security

5.3 Commitment Protocols with Fall-Back Security

Both the pre-processing and online phases require commitment schemes that are maliciously fall-back secure tolerating $\mathcal{Z}_{\mathsf{S}}$. Let $\mathcal{F}_{\mathsf{Com}}$ be an n-party commitment functionality (Figure 9). Note that this functionality proceeds in a 'Commit Phase' and a 'Decommit Phase'. We realize this functionality with an n-party maliciously fall-back secure protocol. For this, we need to define simulators for the case where the adversary does not corrupt the committer C, where it needs to de-commit to the correct value without the knowledge of the input committed

Extractable Commitment with Fall-Back Security

Setting: Let $\mathcal{P} = \{P_i\}_{i\in[n]}$ and $\mathcal{Z}_\mathsf{S} \subset 2^{\mathcal{P}}$. Let $C \in \mathcal{P}$ be the committer with input $x \in \mathcal{X}$. There exist secure point-to-point and broadcast channels for all parties. Let Com_SB (Definition 2.10) be a statistically binding commitment. For the adversary structure $\mathcal{Z}_\mathsf{S}$ let $\mathsf{SS}_{\mathcal{Z}_\mathsf{S}} = (\mathsf{Share}_{\mathcal{Z}_\mathsf{S}}, \mathsf{Recon}_{\mathcal{Z}_\mathsf{S}})$ be a secret-sharing scheme over $\mathcal{X}$. Let Com_SH (Definition 2.11) be a statistically hiding commitment and let SZKAoK be a statistical zero-knowledge argument of knowledge protocol (Definition 2.8) for the statement:

$$\exists x, r \text{ s.t. } c_{\mathsf{Com}_\mathsf{SH}} = \mathsf{Com}_\mathsf{SH}(x; r)$$

For each $k \in [n]$, let ZKPoK be a zero-knowledge proof of knowledge (Definition 2.9) for the statements of the form:

$$\exists [x]_k, r^k_\mathsf{Com} \text{ s.t. } c^k_{\mathsf{Com}_\mathsf{SB}} = \mathsf{Com}_\mathsf{SB}([x]_k; r^k_\mathsf{Com})$$

The extractable commitment protocol Π_ECom operates as follows:

1. **Commit Phase.**
   - C computes shares $\{[x]_k\}_{k\in[n]} \leftarrow \mathsf{Share}_{\mathcal{Z}_\mathsf{S}}(x)$.
   - For each $P_k \in \mathcal{P}$ such that $P_k \neq C$, C samples randomness r^k_Com and computes $c^k_{\mathsf{Com}_\mathsf{SB}} = \mathsf{Com}_\mathsf{SB}([x]_k; r^k_\mathsf{Com})$.
   - For each $P_k \in \mathcal{P}$ such that $P_k \neq C$, C executes ZKPoK as the prover and P_k as the verifier.
   - C samples $r_{\mathsf{Com}_\mathsf{SH}}$ and broadcasts $c_{\mathsf{Com}_\mathsf{SH}} = \mathsf{Com}_\mathsf{SH}(x; r_{\mathsf{Com}_\mathsf{SH}})$.
   - C executes SZKAoK as the prover with all other parties as verifiers.
2. **De-commit Phase.**
   - C sends r^k_Com to each $P_k \in \mathcal{P}$ and broadcasts $(r_{\mathsf{Com}_\mathsf{SH}}, x, \{[x]_k\}_{k\in[n]})$.
   - Each party $P_k \neq C \in \mathcal{P}$, checks if

$$\begin{aligned} x &== \mathsf{Recon}_{\mathcal{Z}_\mathsf{S}}(\{[x]_k\}_{k\in[n]}) \\ c_{\mathsf{Com}_\mathsf{SH}} &== \mathsf{Com}_\mathsf{SH}(x; r_{\mathsf{Com}_\mathsf{SH}}) \\ c^k_{\mathsf{Com}_\mathsf{SB}} &== \mathsf{Com}_\mathsf{SB}([x]_k; r^k_\mathsf{Com}) \end{aligned}$$

   If this holds, accept x as output.

Fig. 11. Extractable Commitment with Fall-Back Security

to and without being able to rewind to the previous phase. As building-blocks, we first describe an n-party commitment protocol (Figure 10) between one committer and $n-1$ viewers that preserves fall-back security. This protocol operates in a commit phase and a decommit phase. It uses as its building-blocks a 2-party statistically binding commitment scheme, a statistically hiding commitment scheme, and an n-party secret-sharing scheme for which a secret cannot be reconstructed for adversary structure $\mathcal{Z}_\mathsf{S}$.

We also require another variant (Figure 11) that is extractable: there exists a PPT extractor that can extract the value committed to while playing the role of one of the viewers. We rely on a statistical zero-knowledge argument of knowledge and a computational zero-knowledge proof of knowledge to achieve extraction. We remark that, since we only need weak extraction, we can achieve this by simply having the committer commit to multiple 2-out-of-2 secret sharings of the message where the receiver challenges the committer to open one out of the two in each of those sharings (e.g., see [PW09]).

n-Party Commitment with Fall-Back Security. Given the two fall-back secure protocols described above, we construct in Figure 12 a protocol that realizes $\mathcal{F}_{\mathsf{Com}}$ (Figure 9) with malicious fall-back security. It allows us to equivocate the committed message in the decommit phase of the simulation. This protocol realizes the required functionality but only for binary inputs in $\{0, 1\}$. For it to work for any input domain, we convert the required input into a binary string and execute multiple executions of this protocol to bit-wise commit to the complete string. In our final construction, we denote by Π_{Com} this commitment protocol for an arbitrary input domain.

5.4 Malicious Fall-Back Secure Protocol for Authenticated Triples

We compile $\Pi_{\mathsf{AuthTriples}}$ as given in Section 5.2 to provide malicious fall-back security tolerating $\mathcal{Z}_{\mathsf{S}}$. This requires putting mechanisms in place that ensure that all the parties execute the protocol $\Pi_{\mathsf{AuthTriples}}$ in a semi-honest manner. This is done in three steps.

1. **Randomness Generation.** For each of the n parties and each of the m executions of Π_S, all the parties execute a coin-tossing in-the-well protocol to sample uniformly the random tape to be used by a party in the execution of Π_S. In each such execution, a designated party receives the random tape and all other parties receive a commitment to this randomness. This is done in a way that preserves malicious fall-back security tolerating $\mathcal{Z}_{\mathsf{S}}$ and uses the commitment and extractable commitment protocols described in Section 5.3. Executing $n \cdot m$ such protocols determines all the randomness to be used in the execution of $\Pi_{\mathsf{AuthTriples}}$, with the exception of the randomness involved in secret-sharing the inputs of each party to create the inputs for each Π_S.
2. **Input Commitment.** Each of the n parties sample their inputs to $\Pi_{\mathsf{AuthTriples}}$ and creates a t-out-of-m threshold-secret-sharing of each of these values as given in Protocol 5.1. The party then uses the fall-back secure extractable commitment protocol to commit to these shares to all other parties.
3. **Consistency Checks.** Intuitively, compiling the semi-honest protocol to malicious security involves first having all the parties commit to all their inputs and randomness and then executing the protocol with the values committed. Owing to the t-out-of-m secret sharing, the semi-honest protocol $\Pi_{\mathsf{AuthTriples}}$ is secure in the presence of up to t failed executions of Π_S. So an adversary corrupting too few of these executions cannot affect security. First, a degree-test needs to be conducted to ensure that the inputs to all the instances of Π_S when looked at collectively are indeed a t-out-of-m secret

n-party protocol realizing $\mathcal{F}_{\mathsf{Com}}$

Setting: Let $\mathcal{P} = \{P_i\}_{i \in [n]}$ and $\mathcal{Z}_{\mathsf{S}} \subset 2^{\mathcal{P}}$. Let $C \in \mathcal{P}$ be the committer with input $x \in \{0,1\}$. There exist secure point-to-point and broadcast channels for all parties. Let Com (Figure 10) be a fall-back secure commitment protocol and ECom (Figure 11) be an extractable fall-back secure commitment protocol tolerating $\mathcal{Z}_{\mathsf{S}}$. Let κ be a security parameter.
The commitment protocol Π_{Com} operates as follows:

1. **Commit Phase.**
   - Each party $P_i \in \mathcal{P}$ such that $P_i \neq C$ samples a string $r_i \in \{0,1\}^{\kappa}$ and executes the 'commit phase' of Com with this as input.
   - For each $j \in [\kappa]$, C samples ν_j and creates a matrix:
   $$M_j = \begin{bmatrix} \nu_j, \nu_j + x \\ \nu_j, \nu_j + x \end{bmatrix}$$
   Execute the 'commit phase' of ECom with each element of M_j as input.
   - For each party $P_i \in \mathcal{P}$ such that $P_i \neq C$ the 'decommit phase' of Com is executed and all parties learn r_i.
   - C can compute $r = \sum_{P_i \neq C \in \mathcal{P}} r_i$. For each $j \in [\kappa]$, For the $r[j]^{th}$ column in M_j, the 'decommit phase' of ECom is executed and all parties learn (ν_j^*, ν_j^{**}).
   - For each $P_i \in \mathcal{P}$ where $P_i \neq C$, for each $j \in [\kappa]$, P_i checks if $\nu_j^* == \nu_j^{**}$. If all checks pass, it accepts $\{M_j\}_{j \in [\kappa]}$ as the commitment to x.
2. **De-commit Phase.**
   - For each $j \in [\kappa]$, C samples $b_j \leftarrow \{0,1\}$ at random and executes the 'decommit phase' of ECom for the commitment in the b_j^{th} row of the $(\neg r[j])^{th}$ column. All parties learn ν_j' and compute $x_j' = \nu_j^* - \nu_j'$.
   - If $\forall j \in [\kappa]$, x_j' is equal to the same value x, accept x as the output.

Fig. 12. Realizing $\mathcal{F}_{\mathsf{Com}}$ with malicious fall-back security

sharing of the inputs of $\Pi_{\mathsf{AuthTriples}}$.
Next, for malicious security, $\frac{t}{2}$ out of the m instances of the Π_S executions are chosen at random, and all the parties need to decommit to the inputs and randomness used for these executions. All parties then check if these have been computed correctly where security in the presence of a malicious adversary is based on the fact that if the adversary deviates in the protocol in more than t of the instances, it would, with overwhelming probability, be detected within the opened executions.

Protocol 5.4 describes the final pre-processing protocol realizing $\mathcal{F}_{\mathsf{AuthTriples}}$. For completeness, Figure 13 describes the online phase protocol.

Protocol 5.4 *Let $\mathcal{P} = \{P_i\}_{i \in [n]}$ and $\mathcal{Z}_{\mathsf{S}} \subset 2^{\mathcal{P}}$. Let f be function and $\vec{x} \in \{0,1\}^n$ be the input such that $P_i \in \mathcal{P}$ has input $x_i \in \vec{x}$. Let $\mathbb{F}$ be a field and $\mathsf{T} \in \mathbb{N}$ be the*

number of multiplication gates in circuit $\mathbf{C}$ *representing* f*. There exist secure point-to-point and broadcast channels for all parties.*
Let κ *be a security parameter,* $t = \kappa$ *and* $m \geq 16t$*. Let* $\Pi_{\mathsf{AuthTriples}}$ *be a semi-honest fall-back secure protocol tolerating* $\mathcal{Z}_{\mathsf{S}}$ *for adaptive adversaries. For each* $j \in [m]$*, let* Π_S^j *denote the* j^{th} *instance of* Π_S *in* $\Pi_{\mathsf{AuthTriples}}$*. Let* Π_{Com} *(Figure 12) be a protocol realizing* $\mathcal{F}_{\mathsf{Com}}$ *with fall-back security for any input domain.*

1. ***Coin Toss Phase.*** *Execute the commit phase of a coin-tossing protocol. Let* $\mathcal{R}_t = \{0,1\}^s$ *where* $s = \frac{t}{2}\log m$*. Let* $\vec{r} \in \mathcal{R}_t$ *be a vector containing* $\frac{t}{2}$ *elements* $\vec{r} = \{j \in \{0,1\}^{\log m}\}$ *indicating the set of* Π_S^j *executions to be opened in the 'Watch-list' step of the 'Check Phase'.*
   - ***Inputs:*** *Each* $P_i \in \mathcal{P}$ *generates randomness* $\vec{r_i} \leftarrow \mathcal{R}_t$*.*
   - ***Protocol*** Π_{coin} *–* ***Commit Phase:***
     - *Sequentially for each* $P_i \in \mathcal{P}$*, execute the* ***commit phase*** *of* Π_{Com} *with* P_i *as the committer and input* $\vec{r_i}$*.*
     - *Let* $\mathbf{View}^{i,k}_{\mathsf{Com},r}$ *denote the view of party* $P_k \neq P_i$ *in this execution.*
   - ***Outputs:*** *Every* $P_i \in \mathcal{P}$ *accepts* $\{\mathbf{View}^{k,i}_{\mathsf{Com},r}\}_{k\in[n]}$*.*

   Let $\mathbb{F}^{2n+6\mathsf{T}+1}$ *be the space of inputs of each party* P_i *to* Π_S^j*. Execute the* ***commit phase*** *of* Π_{coin} *as above:*
   - ***Inputs:*** *Each* $P_i \in \mathcal{P}$ *samples* $\vec{d_i} \leftarrow \mathbb{F}^{2n+6\mathsf{T}+3}$ *where* $\vec{d} = \sum_{i\in[n]} \vec{d_i}$ *becomes the randomness used in the 'Degree Test' step of the 'Check Phase' to verify the correctness of threshold secret-sharing.*
   - ***Outputs:*** *Every* $P_i \in \mathcal{P}$ *accepts* $\{\mathbf{View}^{k,i}_{\mathsf{Com},d}\}_{k\in[n]}$*.*
2. ***Input Commitment Phase.*** $\forall i \in [n]$*,* P_i *commits to its inputs in* $\Pi_{\mathsf{AuthTriples}}$*.*
   - ***Inputs:*** P_i *generates a vector* $\vec{v_i}$ *containing the random elements,*

   $$\Delta_i \leftarrow \mathbb{F}, \left\{r_i^k, \mathsf{MAC}_i(r^k) \leftarrow \mathbb{F}\right\}_{k\in[n]}$$
   $$\left\{a_i^j, b_i^j, c_i^j, \mathsf{MAC}_i(a^j), \mathsf{MAC}_i(b^j), \mathsf{MAC}_i(c^j) \leftarrow \mathbb{F}\right\}_{j\in[\mathsf{T}]}$$

   - ***Protocol*** Π^i_{Com} *–* ***Commit Phase:***
     - P_i *computes for each* $v \in \vec{v_i}$ *the shares* $\{[v]_j\}_{j\in[m]} \leftarrow \mathsf{Share}_{t,m}(v)$*. Let* $\vec{v_i^j} = \{[v]_j\}_{v\in\vec{v_i}}$ *be* P_i*'s inputs to* Π_S^j*.*
     - *Sequentially for each* $j \in [m]$*, execute the* ***commit phase*** *of* Π_{Com} *where* P_i *is the committer with input* $\vec{v_i^j}$*. Let* $\mathbf{View}^{i,k}_{\mathsf{Com},j,v}$ *be the view of each* $P_k \neq P_i$ *in this execution.*
   - ***Outputs:***
     - *Every* $P_k \neq P_i$ *accepts* $\{\mathbf{View}^{k,i}_{\mathsf{Com},j,v}\}_{j\in[m]}$*.*
     - P_i *sets for each* $j \in [m]$*, the input to* Π_S^j *the vector* $\vec{v_i^j}$ *of the form,*

   $$\vec{v_i^j} = \big([\Delta_i]_j, \left\{[r_i^k]_j, [\mathsf{MAC}_i(r^k)]_j\right\}_{k\in[n]}, \left\{[a_i^{j'}]_j, [b_i^{j'}]_j, [c_i^{j'}]_j,\right.$$
   $$\left.[\mathsf{MAC}_i(a^{j'})]_j, [\mathsf{MAC}_i(b^{j'})]_j, [\mathsf{MAC}_i(c^{j'})]_j\right\}_{j'\in[\mathsf{T}]}\big)$$

3. ***Randomness Generation Phase.*** *Sequentially* $\forall i \in [n], j \in [m]$*, the commit phase of a coin-tossing-in-the-well protocol* $\Pi^{i,j}_{\mathsf{CTW}}$ *is executed.*
   Let $P_i \in \mathcal{P}$ *be the receiver,* $\mathcal{R}$ *be the space of random tapes.*

- ***Inputs:*** *Each* $P_k \in \mathcal{P}$ *generates randomness* $r^k_{i,j} \leftarrow \mathcal{R}$.
- ***Protocol*** $\Pi^{i,j}_{\mathsf{CTW}}$ – ***Commit Phase:***
  - *Sequentially for each* $P_k \in \mathcal{P}$, *execute the* ***commit phase*** *of* Π_{Com} *where* P_k *is the committer with input* $r^k_{i,j}$. *Let* $\mathbf{View}^{k,k',i,j}_{\mathsf{Com}}$ *be the view of each* $P_{k'} \neq P_k$.
  - $\forall P_k \neq P_i$, *execute* ***decommit phase*** *of* Π_{Com} *with committer* P_k.
- ***Outputs:***
  - P_i *sets* $r_{i,j} = \sum_{k\in[n]} r^k_{i,j}$ *as its randomness for* Π^j_S.
  - *Every* $P_k \neq P_i$ *accepts* $(\mathbf{View}^{i,k,i,j}_{\mathsf{Com}}, \{r^k_{i,j}\}_{P_k \neq P_i \in \mathcal{P}})$.

4. ***Compute Phase.*** $\forall j \in [m]$, *execute* Π^j_S *(functionality* $\mathcal{F}_{\mathcal{S}}$ *– Figure 8).*
   - ***Inputs:*** *Each* $P_i \in \mathcal{P}$ *uses* $\vec{v^j_i}$ *as the input to* Π^j_S.
   - ***Outputs:*** *Party* $P_i \in \mathcal{P}$ *gets the output vector* $\vec{s^j_i}$.

   Each $P_i \in \mathcal{P}$ *computes* $\vec{s_i} \leftarrow \mathsf{Recon}_{4t,m}(\{\vec{s^j_i}\}_{j\in[m]})$.
5. ***Check Phase.*** *Perform consistency checks on the execution of* $\Pi_{\mathsf{AuthTriples}}$.

   (a) ***Degree Test:***
   - *Each* $P_i \in \mathcal{P}$ *samples* $z_0, z_1 \leftarrow \mathbb{F}$.
     It computes $\vec{z_0} \leftarrow \mathsf{Share}_{t,m}(z_0)$ *and* $\vec{z_1} \leftarrow \mathsf{Share}_{t,m}(z_1)$.
   - *Complete the 'reveal phase' of* Π_{coin} *with domain* $\mathbb{F}^{2n+6\mathsf{T}+1}$:
     - *Sequentially for each* $P_i \in \mathcal{P}$, *execute the* ***decommit phase*** *of* Π_{Com} *where* P_i *is the committer and receive* $\vec{d_i}$.

     Every party $P_i \in \mathcal{P}$ *accepts* $\vec{d} = \sum_{k\in[n]} \vec{d_k} \in \mathbb{F}^{2n+6\mathsf{T}+3}$.
   - $\forall j \in [m]$, $P_i \in \mathcal{P}$ *broadcast* $d_{i,j} = \langle \vec{d}, \vec{s^i_j} \rangle$ *where* $\vec{s^i_j} = (z_0[j], z_1[j], \vec{v^i_j})$.
   - *All parties check* $\exists i \in [n], \mathsf{Recon}_{t,m}(\{d_{i,j}\}_{j\in[m]}) == \perp$ *then ABORT.*

   (b) ***Watch-list:***
   - *Complete the 'reveal phase' of* Π_{coin} *with domain* $\mathcal{R}_t$:
     - *Sequentially for each* $P_i \in \mathcal{P}$, *execute the* ***decommit phase*** *of* Π_{Com} *where* P_i *is the committer and receive* $\vec{r_i}$.
     - *Each party* $P_i \in \mathcal{P}$ *accepts* $\vec{r} = \sum_{k\in[n]} \vec{r_k}$.
   - *For each* $j \in \vec{r}$, $\forall P_i \in \mathcal{P}$, *execute the 'reveal phase' of* $\Pi^{i,j}_{\mathsf{CTW}}$:
     - *Execute the* ***decommit phase*** *of* Π_{Com} *where* P_i *is the committer and receive* $r^i_{i,j}$.
     - *Each* $P_k \neq P_i$ *accepts* $r_{i,j} = \sum_{k'\in[n]} r^{k'}_{i,j}$.
   - *For each* $j \in \vec{r}$, $\forall P_i \in \mathcal{P}$, *reveal the input commitments in* Π^i_{Com}:
     - *Broadcast* $z_0[j]$ *and* $z_1[j]$ *and execute the* ***decommit phase*** *of* Π_{Com} *where* P_i *is the committer and receive* $\vec{v^j_i}$.
   - $\forall j \in \vec{r}$, *check if the view of the execution of* Π^j_S *in the* ***Compute Phase*** *equals the view produced on computing* Π^j_S *with inputs* $\{\vec{v^j_i}\}_{i\in[n]}$ *and randomness* $\{r_{i,j}\}_{i\in[n]}$. *Also for each* $i \in [n]$, *check if* $d_{i,j} == \langle \vec{d}, \vec{s^i_j} \rangle$. *If the check fails, ABORT.*

Theorem 5.5 *For a set of n parties* $\mathcal{P} = \{P_i\}_{i\in[n]}$, *let* $\mathcal{Z}_{\mathsf{S}} \subset 2^{\mathcal{P}}$ *be an adversary structure for unbounded malicious adversaries.*

– *Let $\Pi_{\mathsf{AuthTriples}}$ be an n-party protocol that realizes $\mathcal{F}_{\mathsf{AuthTriples}}$ (Figure 7) with semi-honest fall-back security tolerating $\mathcal{Z}_{\mathsf{S}}$ for adaptive adversaries.*
– *Let Π_{online} be an n-party protocol that can compute $\mathcal{F}$ with malicious fall-back security tolerating $\mathcal{Z}_{\mathsf{S}}$, in the $\mathcal{F}_{\mathsf{AuthTriples}}$-hybrid.*

Then Protocol 5.4 and Figure 13 can compute any function f with malicious fall-back security (Definition 3.2) tolerating $\mathcal{Z}_{\mathsf{S}}$.

The proof for Theorem 5.5 follows from the fact that Π_{online} computes any function in the $\mathcal{F}_{\mathsf{AuthTriples}}$-hybrid with malicious fall-back security. So it remains to prove the following lemma.

Lemma 5.6 *For a set of n parties $\mathcal{P} = \{P_i\}_{i\in[n]}$, let $\mathcal{Z}_{\mathsf{S}} \subset 2^{\mathcal{P}}$ be an adversary structure for unbounded malicious adversaries. Let $\Pi_{\mathsf{AuthTriples}}$ be an n-party protocol that realizes $\mathcal{F}_{\mathsf{AuthTriples}}$ (Figure 7) with semi-honest fall-back security tolerating $\mathcal{Z}_{\mathsf{S}}$ for adaptive adversaries. Then Protocol 5.4 realizes $\mathcal{F}_{\mathsf{AuthTriples}}$ with malicious fall-back security (Definition 3.2) tolerating $\mathcal{Z}_{\mathsf{S}}$.*

The complete proof for this can be found in Appendix A.3. The main theorem requires the semi-honest fall-back secure to be adaptively secure. We argue in Section 5.5 how to relax this requirement and design a protocol without any additional assumptions.

5.5 Removing the Adaptive Security Requirement

We now provide an overview of how to remove the adaptive security requirement on the semi-honest fall-back secure protocol in the malicious compilation.

Let us recall our protocol on a high-level. We emulate a virtual outer protocol where the state of the virtual party is secret shared among the set of parties, and each step of the virtual party in the outer protocol is emulated via an inner protocol. The virtual parties can either be "watched", i.e., the adversary has "full view" of the virtual party, or "unwatched". In particular, for watched parties, the adversary has full information about their input and randomness. In the simulation, the simulator emulates the actions of the honest parties following an honest algorithm for all instances of the inner protocol corresponding to watched virtual parties, but relies on the semi-honest adaptive simulation of the inner protocol to generate the honest parties' messages in the inner protocol executions corresponding to the unwatched virtual parties.

We need the inner protocol simulation to tolerate adaptive corruption to simulate the scenario when the adversary deviates in any inner protocol corresponding to an unwatched virtual party. Specifically, we rely on two properties to simulate when such a deviation occurs: (1) The simulator can identify the precise step when the adversary deviates since it extracts the input and randomness the adversary is supposed to use in emulating all virtual parties at the beginning of the protocol, and (2) When a deviation occurs, the simulator can obtain the real inputs of the honest parties in that inner protocol execution because with high probability the number of unwatched virtual parties on which

Online Protocol using Authenticated Triples Π_{online}

Setting: Let $\mathcal{P} = \{P_i\}_{i\in[n]}$ and $\mathcal{F}$ be an n-party functionality with input $\vec{x} = \{x^i\}_{i\in[n]}$ where each party $P_i \in \mathcal{P}$ has x^i. There exist secure point-to-point and broadcast channels for all parties. Let Π_{Com} (Figure 10) be a commitment protocol with malicious fall-back security with a 'commit' phase and 'de-commit' phase. Each party $P_i \in \mathcal{P}$ has:

$$\vec{s_i} = \{\Delta_i, r^i, \{r_i^k, \mathsf{MAC}_i(r^k)\}_{k\in[n]}, \\ \{a_i^j, b_i^j, c_i^j, \mathsf{MAC}_i(a^j), \mathsf{MAC}_i(b^j), \mathsf{MAC}_i(c^j)\}_{j\in[\mathsf{T}]}\}$$

The parties compute $\mathcal{F}$ step-by-step using the following:

1. **Input Sharing.** All parties create authenticated shares of their inputs.
   - Let x^i be the input of party P_i to be shared.
   - P_i has $r^i, r_i^i, \mathsf{MAC}_i(r^i), \Delta_i \in \vec{s_i}$ and broadcasts $d_i = (x^i - r^i)$.
   - P_i computes $x_i^i = r_i^i + d_i$ and $\mathsf{MAC}_i(x^i) = \mathsf{MAC}_i(r^i) + (\Delta_i \cdot d_i)$.
   - Each party $P_j \neq P_i \in \mathcal{P}$ has $r_j^i, \mathsf{MAC}_j(r^i), \Delta_j \in \vec{s_j}$.
     It computes $x_j^i = r_j^i$ and $\mathsf{MAC}_j(x^i) = \mathsf{MAC}_j(r^i) + (\Delta_j \cdot d_i)$.
2. **Addition Gate Evaluation.** Let the secret inputs be x and y.
   - Each $P_i \in \mathcal{P}$ has $(x_i, \mathsf{MAC}_i(x))$ and $(y_i, \mathsf{MAC}_i(y))$.
   - Each $P_i \in \mathcal{P}$ computes $(x+y)_i = x_i + y_i$
     and $\mathsf{MAC}_i(x+y) = \mathsf{MAC}_i(x) + \mathsf{MAC}_i(y)$.
3. **Adding Public Constant.** Let d be a public constant and x be a secret.
   - Each $P_i \in \mathcal{P}$ has $(x_i, \mathsf{MAC}_i(x))$ and $\Delta_i \in \vec{s_i}$.
   - P_1 computes $(x+d)_1 = x_1 + d$ and $\mathsf{MAC}_1(x+d) = \mathsf{MAC}_1(x) + (\Delta_1 \cdot d)$.
   - Each $P_i \neq P_1 \in \mathcal{P}$ computes $(x+d)_i = x_i$
     and $\mathsf{MAC}_i(x+d) = \mathsf{MAC}_i(x) + (\Delta_i \cdot d)$.
4. **Reconstruction.** Let x be the secret value to be reconstructed.
   - Each $P_i \in \mathcal{P}$ has $(x_i, \mathsf{MAC}_i(x))$ and $\Delta_i \in \vec{s_i}$.
   - Each $P_i \in \mathcal{P}$ broadcasts x_i and then computes $x' = \sum_{k\in[n]} x_k$.
   - Each $P_i \in \mathcal{P}$ computes $d_i = \Delta_i \cdot x' - \mathsf{MAC}_i(x)$ and **commit** to it.
   - Each $P_i \in \mathcal{P}$ **decommit** to d_i and checks if $\sum_{k\in[n]} d_k == 0$. If this is true, accept $x' = x$.
5. **Multiplication Gate Evaluation.** Let the secret inputs be x and y.
   - Each $P_i \in \mathcal{P}$ has $(x_i, \mathsf{MAC}_i(x))$ and $(y_i, \mathsf{MAC}_i(y))$.
   - Each $P_i \in \mathcal{P}$ also has $\Delta_i, a_i, b_i, c_i, \mathsf{MAC}_i(a), \mathsf{MAC}_i(b), \mathsf{MAC}_i(c) \in \vec{s_i}$.
   - Each $P_i \in \mathcal{P}$ follows the steps in '**Addition Gate Evaluation**' to compute $e_i = (x-a)_i$ and $\mathsf{MAC}_i(e)$; and $d_i = (y-b)_i$ and $\mathsf{MAC}_i(d)$.
   - Each $P_i \in \mathcal{P}$ does '**Reconstruction**' to get e and d, and compute ed.
   - Each $P_i \neq P_1 \in \mathcal{P}$ computes $(xy)_i = c_i + e \cdot b_i + d \cdot a_i$
     and $\mathsf{MAC}_i(xy) = \mathsf{MAC}_i(c) + e \cdot \mathsf{MAC}_i(b) + d \cdot \mathsf{MAC}_i(a) + ed\Delta_i$
   - Each P_1 computes $(xy)_1 = c_1 + e \cdot b_1 + d \cdot a_1 + ed$
     and $\mathsf{MAC}_1(xy) = \mathsf{MAC}_1(c) + e \cdot \mathsf{MAC}_1(b) + d \cdot \mathsf{MAC}_1(a) + ed\Delta_1$

Fig. 13. Online Protocol using Authenticated Triples

the adversary deviates is bounded and we can rely on the adaptive simulation of the outer protocol to generate their inputs and randomness. Therefore, when the deviation occurs, the simulator obtains the actual inputs for the honest parties for that inner protocol execution and provides it to the adaptive simulator to complete the simulation.

[IPS08] proved the security of their protocol, assuming honest parties can erase data. Our main observation is that it suffices that the inner protocol admits adaptive simulation with erasures, but the honest parties in the real world do not have to erase anything. Before we argue this, we briefly explain how our inner protocol, namely the semi-honest fall-back secure protocol, can be modified slightly to admit adaptive simulation with erasures.

Adaptive erasures property of the inner protocol. Recall that the warmup protocol proceeds in two phases. First, a designated subset of $n-1$ parties jointly build a garbled circuit. Then the designated nth party executes multiple oblivious transfers as a receiver (that is secure unconditionally against passive senders) with the remaining $n-1$ parties as senders. We modify the warmup protocol by having the $n-1$ parties execute sufficiently many oblivious-transfers on random inputs with the n^{th} party in the first phase and then use a reduction to random OTs in the second phase when executing the actual OTs. Observe that the first phase is input independent (for both the garbling and random OT). Therefore, we will have all the parties erase their local randomness used in the first phase and only carry forward the garbling result and the inputs and outputs of the random OTs. This variant of the inner protocol admits adaptive simulation with erasures. Observing that semi-honest protocols of Yao's garbled circuits and GMW have been previously observed to be adaptively secure in the presence of erasures [IPS08], we briefly explain why our protocol satisfies adaptive simulation with erasures as well. In the first phase, the simulator simulates the honest parties honestly; thereby, any corruption can be simulated perfectly. In the second phase, any adaptive corruption will require the simulator only to reveal the garbled circuit and the random OT inputs and outputs, but not the randomness used to generate them as they have been erased.

Honest parties do not have to erase in the real world. Finally, we argue why the honest parties do not actually have to erase their data in any of the executions of the warmup protocol in our main protocol. The idea here is that when the adversary deviates in an inner protocol instance, it does not actually (adaptively) corrupt any honest party in the real world. So it does not obtain the view of any honest party. Hence, we *only* need a mechanism for the simulator to continue simulating the honest parties and not have to share the internal state of the honest parties with the adversary (which is required if the adversary corrupted the honest parties). Our main observation is that we can rely on the adaptive simulation assuming erasures for such a mechanism.

More precisely, in the simulation, when the adversary deviates in an instance of the internal protocol, the main simulator invokes the adaptive simulation of the outer protocol to obtain the partial view of the corresponding virtual party. From this view, the main simulator can obtain the inputs of the honest parties within the inner protocol instance on which the adversary deviated. We recall here that each inner protocol instance corresponds to some next-message computation of a virtual party, and the parties hold secret-shares of the state of that virtual party. Thus, determining the inputs and outputs of the honest parties in an inner protocol instance becomes sampling shares for the honest parties that reconstruct to the right value (i.e., the inputs and outputs inferred from the view of the virtual party). Given the inputs and outputs of the honest parties in the inner protocol instance, the main simulator emulates an adaptive corruption of the honest parties in that instance. Now, we rely on the adaptive simulator of the inner protocol (with erasures), where the main simulator provides the inputs and outputs of the honest parties to provide a view of the honest parties consistent with the simulation thus far. From there on, the main simulator uses the honest algorithm to generate the honest parties' messages for the adaptively corrupted instance. We emphasize that the main simulator obtains the view of the honest parties but does not share that with the adversary.

To argue security, we observe that any adversary $\mathcal{A}$ that can distinguish the real world from this simulation can be transformed into another adversary $\mathcal{B}$ that can break the adaptive security with erasures property of the warmup/inner protocol. This is because $\mathcal{B}$ can emulate the adversary $\mathcal{A}$, and whenever a deviation occurs, it will adaptively corrupt the honest parties and receive their internal state of the honest parties but ignore it as $\mathcal{A}$ does not need it. If $\mathcal{A}$ breaks the security in an execution of our protocol, it will imply an instance of the warmup protocol where the $\mathcal{B}$ breaks the adaptive security with erasures property.

Acknowledgments

Anasuya Acharya and Carmit Hazay are supported by ISF grant No. 1316/18. Carmit Hazay is also supported by the Algorand Centres of Excellence programme managed by Algorand Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Algorand Foundation. The fourth author was supported by a JPMorgan Chase Faculty Research Award, Technology, and Humanity Fund from the McCourt School of Public Policy at Georgetown University, and a Google Research Award.

References

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

BHR12. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *ACM CCS*, pages 784–796, 2012.

BK95. G. R. Blakley and Gregory Kabatianskii. General perfect secret sharing schemes. In *CRYPTO 1995*, pages 367–371. Springer, 1995.

BLO16. Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the internet. In *ACM SIGSAC*, pages 578–590, 2016.

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

BPS22. Saikrishna Badrinarayanan, Sikhar Patranabis, and Pratik Sarkar. Statistical security in two-party computation revisited. In *TCC 2022*, pages 181–210, 2022.

CDvdG87. David Chaum, Ivan Damgård, and Jeroen van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *CRYPTO 1987*, pages 87–119, 1987.

Cha89. David Chaum. The spymasters double-agent problem: Multiparty computations secure unconditionally from minorities and cryptographically from majorities. In *CRYPTO 1989*, pages 591–602, 1989.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO 2012*, pages 643–662, 2012.

EGL82. Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. In *CRYPTO 1982*, pages 205–210, 1982.

GMPP16. Sanjam Garg, Pratyay Mukherjee, Omkant Pandey, and Antigoni Polychroniadou. The exact round complexity of secure computation. In *EUROCRYPT 2016*, pages 448–476, 2016.

HM97. Martin Hirt and Ueli M. Maurer. Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract). In *PODC 1997*, pages 25–34, 1997.

HVW20. Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, and Mor Weiss. The price of active security in cryptographic protocols. In *EUROCRYPT 2020*, pages 184–215, 2020.

IKK$^+$11. Yuval Ishai, Jonathan Katz, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. On achieving the "best of both worlds" in secure multiparty computation. *SIAM J. Comput.*, 40(1):122–141, 2011.

IPS08. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO 2008*, pages 572–591, 2008.

Kat07. Jonathan Katz. On achieving the "best of both worlds" in secure multiparty computation. In *ACM STOC 2007*, pages 11–20, 2007.

KM20. Dakshita Khurana and Muhammad Haris Mughees. On statistical security in two-party computation. In *TCC 2020*, pages 532–561, 2020.

KO04. Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In *CRYPTO 2004*, pages 335–354, 2004.

LPSY15. Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In *CRYPTO*, pages 319–338, 2015.

PW09. Rafael Pass and Hoeteck Wee. Black-box constructions of two-party protocols from one-way functions. In *TCC 2009*, pages 403–418, 2009.

Sha79. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

A Omitted Proofs

A.1 Proof of Lemma 3.4

Proof. The proof of this lemma follows in two parts. First we show that there exists a PPT simulator Sim_S that for any subset of parties $Z \in \mathcal{Z}_\mathsf{S}$ can output a view whose distribution is statistically close to the view in the real protocol in the plain model. Next, we go on to show that there exists a PPT simulator Sim_C that for any subset of parties $Z \in \mathcal{Z}_\mathsf{C} = 2^{\mathcal{P}}$ can output a view that is computationally indistinguishable from the distribution of the view in the real protocol.

In order to examine the security of Figure 4 in the plain model, it becomes necessary to describe the properties of its various building blocks:

1. $\mathcal{F}_\mathsf{GS}$ used in Figure 4 is replaced by protocol Π_GS [BLO16] that computes a multi-party garbling and decoding information for the function f. Since $\mathcal{F}_\mathsf{GS}$ does not take the function inputs $\vec{x}$, the execution of Π_GS is also independent of it. The protocol requires each of the garblers in $\mathcal{P}_\mathsf{N}$ to sample randomness and create the inputs that are listed in the description of $\mathcal{F}_\mathsf{GS}$. Then the garblers execute a protocol that generates output of the functionality and this is also given to the evaluator P_n. The complete protocol Π_GS is secure against a semi-honest PPT adversary corrupting any subset Z of $n-1$ parties in $\mathcal{P}$. Therefore, there exists a PPT simulator $\mathsf{Sim}_\mathsf{C}^\mathsf{GS}$ that takes the inputs and outputs of all the corrupt parties and produces a view of the protocol that is computationally indistinguishable from the real view:
$$\begin{aligned}\hat{\mathbf{View}}_\mathsf{GS}^{Z} \leftarrow \mathsf{Sim}_\mathsf{C}^\mathsf{GS}(1^\kappa, &\{\lambda_w^i, k_{w,0}^i, k_{w,1}^i\}_{w\in[n+q],P_i\in Z},\\ &\{F_{a,b}^{i,j,g}\}_{g\in[q],j\in[\mathsf{N}],(a,b)\in\{0,1\}^2,P_i\in Z},\\ &\{G_{a,b}^{j}\}_{j\in[\mathsf{N}],g\in[q],(a,b)\in\{0,1\}^2},\\ &\{\lambda_w\}_{w\in[n+q-m,n+q]},\{\lambda_i\}_{P_i\in Z})\end{aligned}$$
Here, $\hat{\mathbf{View}}_\mathsf{GS}^{Z}$ denotes the view of $Z \subset \mathcal{P}$ output by simulator $\mathsf{Sim}_\mathsf{C}^\mathsf{GS}$.
2. $\mathcal{F}_\mathsf{OT}$ used in Figure 4 is replaced by a protocol Π_OT [EGL82] that computes 2-party oblivious transfer. Π_OT is a protocol that is secure in the presence of a computationally unbounded semi-honest adversary corrupting the OT sender. That is, there exists a simulator $\mathsf{Sim}_\mathsf{S}^\mathsf{OT}$ that takes the inputs of the sender and produces a view of the protocol that is statistically close to that of the real view:
$$\hat{\mathbf{View}}_\mathsf{OT}^{S} \leftarrow \mathsf{Sim}_\mathsf{S}^\mathsf{OT}(1^s, s_0, s_1)$$
Π_OT is also secure in the presence of a PPT semi-honest adversary corrupting the OT receiver. That is, there exists a PPT simulator $\mathsf{Sim}_\mathsf{C}^\mathsf{OT}$ that takes the input and output of the receiver and produces a view of the protocol that is computationally indistinguishable from the real view:
$$\hat{\mathbf{View}}_\mathsf{OT}^{R} \leftarrow \mathsf{Sim}_\mathsf{C}^\mathsf{OT}(1^\kappa, b, s_b)$$

Given the sub-protocols Π_{GS} and Π_{OT} as above, we are now ready to prove our theorem in the plain model.

Let us first consider the case of statistical security against any $Z \in \mathcal{Z}_{\mathsf{S}}$ where $Z \subseteq \{P_i\}_{i \in [n-1]}$. In the plain model, the PPT simulator $\mathsf{Sim}_{\mathsf{S}}$ for Figure 4, making black-box calls to $\mathsf{Sim}_{\mathsf{S}}^{\mathsf{OT}}$, operates as follows:

1. $\mathsf{Sim}_{\mathsf{S}}$ has the input of all the corrupt parties $\{x_i\}_{P_i \in Z}$ and the function output $f(x)$. Let $\mathcal{R}$ be the domain of each party's randomness. $\mathsf{Sim}_{\mathsf{S}}$ samples randomness $\{r_i \leftarrow \mathcal{R}\}_{P_i \in Z}$ for all corrupt parties.
2. In the **Garbling Phase** of Figure 4, $\mathsf{Sim}_{\mathsf{S}}$ samples $\{r_i \leftarrow \mathcal{R}\}_{P_i \in [n]-Z}$ on behalf of all the honest garblers. It executes Π_{GS} as in the real execution with inputs derived from $\vec{r} = \{r_i\}_{P_i \in [n]}$. Let $\mathbf{View}_{\mathsf{GS}}^{Z}$ denote the view of Z in this execution of Π_{GS}.
3. In the **OT Phase** of Figure 4, first $\mathsf{Sim}_{\mathsf{S}}$ computes $\{\Lambda_i\}_{P_i \in Z}$ for the corrupt parties. It then samples $\Lambda_i \leftarrow \{0,1\}$ uniformly at random for each honest party. Next, for each $i \in [n], P_j \in Z$, $\mathsf{Sim}_{\mathsf{S}}$ makes black-box calls to $\mathsf{Sim}_{\mathsf{S}}^{\mathsf{OT}}$, obtaining the views:
$$\forall i \in [n], P_j \in Z, \hat{\mathbf{View}}_{\mathsf{OT},i,j}^{S} \leftarrow \mathsf{Sim}_{\mathsf{S}}^{\mathsf{OT}}(1^s, k_{i,0}^{j}, k_{i,1}^{j})$$
4. Finally, in the **Evaluation Phase**, $\mathsf{Sim}_{\mathsf{S}}$ sets $f(\vec{x})$ as the output.

Let κ be the computational security parameter. The view of the adversary in the above execution of $\mathsf{Sim}_{\mathsf{S}}$ is distributed as,

$$\left\{ \{x_i\}_{P_i \in Z}, f(\vec{x}), \{r_i\}_{P_i \in Z}, \mathbf{View}_{\mathsf{GS}}^{Z}, \{\hat{\mathbf{View}}_{\mathsf{OT},i,j}^{S}\}_{i \in [n], P_j \in Z} \right\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$$

Note that this view differs from the real view only in the view of the oblivious transfer. Let $d = |Z|$. For each $i \in [n], P_j \in Z$, let $\mathbf{View}_{\mathsf{OT},i,j}^{S}$ be the view of oblivious transfer in the real OT execution where P_j is the sender and P_n is the receiver for the choice bit Λ_i. In order to show that the simulated view and the real view of the protocol is statistically close, consider $dn+1$ hybrids of the form:

– Hybrid $\mathsf{H}_{0,0}$. This is the distribution of the output of $\mathsf{Sim}_{\mathsf{S}}$. This is identical to the view in the real execution up to before the OT executions.
$$\mathsf{H}_{0,0} = \left\{ \{x_i\}_{P_i \in Z}, f(\vec{x}), \{r_i\}_{P_i \in Z}, \mathbf{View}_{\mathsf{GS}}^{Z}, \{\hat{\mathbf{View}}_{\mathsf{OT},i,j}^{S}\}_{i \in [n], P_j \in Z} \right\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$$
– Hybrid $\mathsf{H}_{i,j}$. For $i \in [n]$ and corrupted party $P_j \in Z$, this hybrid experiment contains the real execution of Protocol 4 up to the OT execution where P_j is the sender and Λ_i is the input bit of P_n. The rest of the OT execution

views are created as in the simulation using calls to $\mathsf{Sim}_\mathsf{S}^\mathsf{OT}$.

$$\mathsf{H}_{i,j} = \Big\{ \{x_{i'}\}_{P_{i'} \in Z}, f(\vec{x}), \{r_{i'}\}_{P_{i'} \in Z}, \mathbf{View}_\mathsf{GS}^Z, \{\mathbf{View}_{\mathsf{OT},i',j'}^S\}_{i' \in [i-1], (i'=i, j' \leq j)}, \{\hat{\mathbf{View}}_{\mathsf{OT},i',j'}^S\}_{i' > i, (i'=i, j'>j)} \Big\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$$

– Hybrid $\mathsf{H}_{n,d}$. This has the distribution of the view in the real execution of Protocol 4.

$$\mathsf{H}_{n,d} = \Big\{ \{x_i\}_{P_i \in Z}, f(\vec{x}), \{r_i\}_{P_i \in Z}, \mathbf{View}_\mathsf{GS}^Z, \{\mathbf{View}_{\mathsf{OT},i,j}^S\}_{i \in [n], P_j \in Z} \Big\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$$

Claim. Assuming that the OT protocol Π_OT is secure against an unbounded semi-honest adversary corrupting the sender, the view in the hybrid distribution $\mathsf{H}_{i,j}$ is statistically indistinguishable from the view in $\mathsf{H}_{i,j+1}$.

Proof. Note that each adjacent pair of hybrids $\mathsf{H}_{i,j}$ and $\mathsf{H}_{i,j+1}$ (similarly, $\mathsf{H}_{0,0}$ and $\mathsf{H}_{1,1}$, and each $\mathsf{H}_{i,d}$ and $\mathsf{H}_{i+1,1}$) differ only in the view of one OT execution. In $\mathsf{H}_{i,j}$, the view of the OT execution $\hat{\mathbf{View}}_{\mathsf{OT},i,j}^S$ is the simulated OT view output by $\mathsf{Sim}_\mathsf{S}^\mathsf{OT}$. In $\mathsf{H}_{i,j+1}$, this is $\mathbf{View}_{\mathsf{OT},i,j}^S$, as in the real OT execution. The rest of the hybrid is created and distributed the same way. We already have that the view produced by $\mathsf{Sim}_\mathsf{S}^\mathsf{OT}$ is statistically close to the real view of OT. Let ϵ be the statistical difference between the distributions $\{\hat{\mathbf{View}}_{\mathsf{OT},i,j}^S\}$ and $\{\mathbf{View}_{\mathsf{OT},i,j}^S\}$ of the simulated and real OT execution. Then the statistical difference between the hybrid distributions $\mathsf{H}_{i,j}$ and $\mathsf{H}_{i,j+1}$ is no more than ϵ.

It follows from the triangle inequality of statistical differences that the difference between the distribution of the simulated view $\mathsf{H}_{0,0}$ output by Sim_S and the real view $\mathsf{H}_{n,d}$ in Figure 4 is $\leq nd\epsilon$. Therefore, the distributions of the real and simulated views are statistically close.

It remains to handle the case of security against any $Z \in \mathcal{Z}_\mathsf{C}$. Here a PPT semi-honest adversary has in its view $Z \subseteq \mathcal{P}$. The difference between this case and the previous case is that the evaluator of the garbling, P_n, can also be corrupted by the adversary. As such the PPT simulator Sim_C needs to give the adversary a garbled circuit that evaluates to the correct function output, without knowing the inputs of the honest parties to this function. The simulator Sim_C for Figure 4 in the plain model, making black-box calls to $\mathsf{Sim}_\mathsf{C}^\mathsf{GS}$ and $\mathsf{Sim}_\mathsf{C}^\mathsf{OT}$, works as follows:

1. Sim_C has as input the input of all the corrupt parties $\{x_i\}_{P_i \in Z}$ and the function output $f(\vec{x})$. It samples randomness $\{r_i\}_{P_i \in Z}$ for all corrupt parties. If P_n is not corrupted, Sim_C behaves the same as Sim_S.
2. Otherwise, in the **Garbling Phase** of Figure 4, Sim_C first samples randomness on behalf of all the honest garblers and creates a garbling that always

outputs $f(\vec{x})$. This is done by computing,

$$G^{j*}_{a,b} = \left(\bigoplus_{i=1}^{\mathsf{N}} F^{i,j,g}_{a,b}\right) \oplus (\lambda_w || k^j_{w,0}) \qquad \forall j \in [\mathsf{N}], g \in [q], (a,b) \in \{0,1\}^2$$
$$\lambda^*_w = \Lambda_w \oplus y_{w'} \qquad \forall y_{w'} \in f(\vec{x}), w = n + q - m + w'$$

That is, for a garbled gate $g \in [q]$ and $j \in [\mathsf{N}]$, all the 4 ciphertexts mask the same label $k^j_{w,0}$. Finally, for each circuit output wire, the apparent value Λ_w is mapped to the output bit $y \in f(\vec{x})$ by setting $\lambda_w = \Lambda_w \oplus y$.
Next, $\mathsf{Sim_C}$ makes a black-box call to $\mathsf{Sim}^{\mathsf{GS}}_{\mathsf{C}}$ with the inputs of the corrupted parties and the above garbling:

$$\begin{aligned}\hat{\mathbf{View}}^Z_{\mathsf{GS}} \leftarrow \mathsf{Sim}^{\mathsf{GS}}_{\mathsf{C}}(1^\kappa, &\{\lambda^i_w, k^i_{w,0}, k^i_{w,1}\}_{w\in[n+q], P_i \in Z},\\ &\{F^{i,j,g}_{a,b}\}_{g\in[q], j\in[\mathsf{N}], (a,b)\in\{0,1\}^2, P_i\in Z},\\ &\{G^{j*}_{a,b}\}_{j\in[\mathsf{N}], g\in[q], (a,b)\in\{0,1\}^2},\\ &\{\lambda^*_w\}_{w\in[n+q-m,n+q]}, \{\lambda_i\}_{P_i\in Z})\end{aligned}$$

Such a simulator exists [BLO16] and produces a view $\hat{\mathbf{View}}^Z_{\mathsf{GS}}$ that is computationally indistinguishable from the view of Z in a real execution of Π_{GS}.

3. In the **OT Phase** of Figure 4, first $\mathsf{Sim_C}$ samples for each honest party $P_i \in \mathcal{P} - Z$, a random bit $\Lambda_i \leftarrow \{0,1\}$. Next, for each $i \in [n], P_j \in \mathcal{P} - Z$, $\mathsf{Sim_C}$ makes black-box calls to $\mathsf{Sim}^{\mathsf{OT}}_{\mathsf{C}}$, obtaining the views:

$$\forall i \in [n], P_j \in \mathcal{P} - Z, \hat{\mathbf{View}}^R_{\mathsf{OT},i,j} \leftarrow \mathsf{Sim}^{\mathsf{OT}}_{\mathsf{C}}(1^\kappa, \Lambda_i, k^j_{i,\Lambda_i})$$

4. Finally, in the **Evaluation Phase**, $\mathsf{Sim_C}$ accepts $f(\vec{x})$ as the result of garbling evaluation.

Let κ be a computational security parameter and $\vec{r}$ be the randomness sampled for all the corrupt parties and internally in the garbling simulator $\mathsf{Sim}^{\mathsf{GS}}_{\mathsf{C}}$. The view of the adversary in the above execution of $\mathsf{Sim_C}$ is distributed as,

$$\left\{\{x_i\}_{P_i\in Z}, f(\vec{x}), \{r_i\}_{P_i\in Z}, \hat{\mathbf{View}}^Z_{\mathsf{GS}}, \{\hat{\mathbf{View}}^R_{\mathsf{OT},i,j}\}_{i\in[n], P_j\notin Z}\right\}_{\kappa\in\mathbb{N}, \vec{r}\in\mathcal{R}^n}$$

It remains to argue that the above distribution is computationally indistinguishable from that of the real view of the adversary in Figure 4. This view differs from the real view in that the garbling view $\hat{\mathbf{View}}^Z_{\mathsf{GS}}$ and the view of the OT executions $\{\hat{\mathbf{View}}^R_{\mathsf{OT},i,j}\}_{i\in[n], P_j\notin Z}$ are created differently. Let d be the number of corrupt parties. In order to show that the distribution of the output of $\mathsf{Sim_C}$ is indistinguishable from the real view, consider the following set of $n(n-d)+2$ hybrids:

– Hybrid H_0. This is the distribution of the output of $\mathsf{Sim_C}$.

$$\mathsf{H}_0 = \Big\{ \{x_{i'}\}_{P_{i'} \in Z}, f(\vec{x}), \{r_{i'}\}_{P_{i'} \in Z}, \hat{\mathbf{View}}^{Z}_{\mathsf{GS}}, \{\hat{\mathbf{View}}^{R}_{\mathsf{OT},i',j'}\}_{i' \in [n], P_{j'} \notin Z} \Big\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$$

– Hybrid H_1. This is an intermediate distribution where the view is generated by first executing the real Protocol 4 up to the end of Π_{GS}. Then the views of the OT protocols are generated as in the simulation $\mathsf{Sim_C}$.

$$\mathsf{H}_1 = \Big\{ \{x_{i'}\}_{P_{i'} \in Z}, f(\vec{x}), \{r_{i'}\}_{P_{i'} \in Z}, \mathbf{View}^{Z}_{\mathsf{GS}}, \{\hat{\mathbf{View}}^{R}_{\mathsf{OT},i',j'}\}_{i' \in [n], P_{j'} \notin Z} \Big\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$$

This distribution differs from H_0 only in that the view $\mathbf{View}^{Z}_{\mathsf{GS}}$ here is derived from a real execution of Π_{GS}.

– Hybrid $\mathsf{H}_{i,j}$. For $i \in [n]$ and honest party $P_j \notin Z$, this hybrid experiment contains the view of the real execution of Protocol 4 up to the OT execution where P_j is the sender and Λ_i is the input bit of P_n. The rest of the OT execution views are created as in the simulation using calls to $\mathsf{Sim}^{\mathsf{OT}}_{\mathsf{C}}$.

$$\mathsf{H}_{i,j} = \Big\{ \{x_{i'}\}_{P_{i'} \in Z}, f(\vec{x}), \{r_{i'}\}_{P_{i'} \in Z}, \mathbf{View}^{Z}_{\mathsf{GS}}, \{\mathbf{View}^{R}_{\mathsf{OT},i',j'}\}_{i'<i,(i'=i,j' \le j)}, \{\hat{\mathbf{View}}^{R}_{\mathsf{OT},i',j'}\}_{i'>i,(i'=i,j'>j)} \Big\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$$

– Hybrid $\mathsf{H}_{n,(n-d)}$. This has the distribution of the view in the real execution of Protocol 4.

$$\mathsf{H}_{n,(n-d)} = \Big\{ \{x_{i'}\}_{P_{i'} \in Z}, f(\vec{x}), \{r_{i'}\}_{P_{i'} \in Z}, \mathbf{View}^{Z}_{\mathsf{GS}}, \{\mathbf{View}^{R}_{\mathsf{OT},i',j'}\}_{i' \in [n], P_{j'} \notin Z} \Big\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$$

Claim. Assuming that the protocol Π_{GS} is secure against a PPT semi-honest adversary corrupting any subset $Z \subset \mathcal{P}$ of the parties, the view in the hybrid distribution H_0 is computationally indistinguishable from that in H_1.

Proof. We show that if there existed a PPT distinguisher D that can distinguish between H_0 and H_1 with non-negligible advantage ϵ, then D can be used in a black-box way by a PPT adversary $\mathcal{A}_{\mathsf{GS}}$ that distinguishes between the distributions $\{\mathbf{View}^{Z}_{\mathsf{GS}}\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$ from the real execution of Π_{GS} and $\{\hat{\mathbf{View}}^{Z}_{\mathsf{GS}}\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$ from the output of $\mathsf{Sim}^{\mathsf{GS}}_{\mathsf{C}}$. The adversary works as follows:

– The adversary $\mathcal{A}_{\mathsf{GS}}$ has the values $(\vec{x}, f(\vec{x}))$ and samples the randomness $\vec{r} \leftarrow \mathcal{R}^n$ for all parties.
– $\mathcal{A}_{\mathsf{GS}}$ sends $(f(\vec{x}), \vec{r})$ to the challenger $\mathcal{C}$ that internally samples a bit b.
– If $b = 0$, $\mathcal{C}$ computes,

$$G^{j*}_{a,b} = \left(\bigoplus_{i=1}^{\mathsf{N}} F^{i,j,g}_{a,b}\right) \oplus (\lambda_w || k^j_{w,0}) \qquad \forall j \in [\mathsf{N}], g \in [q], (a,b) \in \{0,1\}^2$$
$$\lambda^*_w = \Lambda_w \oplus y_{w'} \qquad \forall y_{w'} \in f(\vec{x}), w = n + q - m + w'$$

$$\begin{aligned}\mathbf{View}_{\mathsf{GS}} \leftarrow \mathsf{Sim}^{\mathsf{GS}}_{\mathsf{C}}(1^\kappa, &\{\lambda^i_w, k^i_{w,0}, k^i_{w,1}\}_{w \in [n+q], P_i \in Z}, \\ &\{F^{i,j,g}_{a,b}\}_{g \in [q], j \in [\mathsf{N}], (a,b) \in \{0,1\}^2, P_i \in Z}, \\ &\{G^{j*}_{a,b}\}_{j \in [\mathsf{N}], g \in [q], (a,b) \in \{0,1\}^2}, \\ &\{\lambda^*_w\}_{w \in [n+q-m, n+q]}, \{\lambda_i\}_{P_i \in Z})\end{aligned}$$

Otherwise, $\mathcal{C}$ computes $\mathbf{View}_{\mathsf{GS}}$ to be the view of the parties in Z in the real execution of Π_{GS}.
– $\mathcal{C}$ gives $\mathbf{View}_{\mathsf{GS}}$ to $\mathcal{A}_{\mathsf{GS}}$ and, with this, $\mathcal{A}_{\mathsf{GS}}$ computes the rest of the view $\mathbf{View}$ as in $\mathsf{Sim}_{\mathsf{C}}$. Note that if $b = 0$, $\mathbf{View}$ is distributed as in H_0 and if $b = 1$, it distributed as in H_1.
– Finally, $\mathcal{A}_{\mathsf{GS}}$ passes $\mathbf{View}$ on to D and outputs whatever D outputs.

In the above strategy, $\mathcal{A}_{\mathsf{GS}}$ has the same distinguishing advantage as D, which is non-negligible. However, since the protocol Π_{GS} is secure against a PPT semi-honest adversary corrupting any subset $Z \subset \mathcal{P}$ of the parties, no such $\mathcal{A}_{\mathsf{GS}}$ can exist [LPSY15, BLO16], and it follows that no such D can exist. So the distributions H_0 and H_1 are computationally indistinguishable.

Claim. Assuming that the OT protocol Π_{OT} is secure against a PPT semi-honest adversary corrupting the receiver, the view in the hybrid distribution $\mathsf{H}_{i,j}$ is computationally indistinguishable from the view in $\mathsf{H}_{i,j-1}$.

Proof. Note that each adjacent pair of hybrids $\mathsf{H}_{i,j}$ and $\mathsf{H}_{i,j-1}$ (similarly, H_1 and $\mathsf{H}_{1,1}$, and each $\mathsf{H}_{i,(n-d)}$ and $\mathsf{H}_{i+1,1}$) differ only in the view of one OT execution. In $\mathsf{H}_{i,j-1}$, the view of the OT execution $\hat{\mathbf{View}}^R_{\mathsf{OT},i,j}$ is the simulated OT view output by $\mathsf{Sim}^{\mathsf{OT}}_{\mathsf{C}}$. In $\mathsf{H}_{i,j}$, this is $\mathbf{View}^R_{\mathsf{OT},i,j}$, as in the real OT execution. The rest of the hybrid is created and distributed the same way.

We show that if there existed a PPT distinguisher D that can distinguish between $\mathsf{H}_{i,j}$ and $\mathsf{H}_{i,j-1}$ with non-negligible advantage ϵ, then D can be used in a black-box way by a PPT adversary $\mathcal{A}_{\mathsf{OT}}$ that distinguishes between the distributions $\{\mathbf{View}^R_{\mathsf{OT},i,j}\}_{\kappa \in \mathbb{N}, r \in \mathcal{R}}$ from the execution of Π_{OT} and $\{\hat{\mathbf{View}}^R_{\mathsf{OT},i,j}\}_{\kappa \in \mathbb{N}, r \in \mathcal{R}}$ from the output of $\mathsf{Sim}^{\mathsf{OT}}_{\mathsf{C}}$. The adversary works as follows:

– $\mathcal{A}_{\mathsf{OT}}$ has the indices i, j and the values $(\vec{x}, f(\vec{x}))$. It samples randomness $\vec{r} \leftarrow \mathcal{R}^n$ and computes $\mathbf{View}^Z_{\mathsf{GS}}$ as in a real execution of the protocol Π_{GS}.

- Then, for all OT instances (i', j') where $i' < i$ or $i' = i$ and $j' < j$, it creates $\mathbf{View}^R_{\mathsf{OT},i',j'}$ as in the real OT execution. For all OT instances (i', j') where $i' > i$ or $i' = i$ and $j' > j$, it creates $\hat{\mathbf{View}}^R_{\mathsf{OT},i',j'}$ as in the simulated OT execution.
- Corresponding to the $(i, j)^{th}$ OT execution, $\mathcal{A}_{\mathsf{OT}}$ sends inputs $k^j_{i,0}, k^j_{i,1}$ and Λ_i to the challenger $\mathcal{C}$.
- $\mathcal{C}$ internally samples a bit b and if $b = 0$, it computes,

$$\mathbf{View}_{\mathsf{OT}} \leftarrow \mathsf{Sim}^{\mathsf{OT}}_{\mathsf{C}}(1^\kappa, \Lambda_i, k^j_{i,\Lambda_i})$$

  Otherwise it computes $\mathbf{View}_{\mathsf{OT}}$ to be the view of the receiver in the real execution of Π_{OT}.
- $\mathcal{C}$ gives $\mathbf{View}_{\mathsf{OT}}$ to $\mathcal{A}_{\mathsf{OT}}$ and, with this, $\mathcal{A}_{\mathsf{OT}}$ completes the view $\mathbf{View}$ to be input to D. Note that if $b = 0$, $\mathbf{View}$ is distributed as in $\mathsf{H}_{i,j-1}$ and if $b = 1$, it is distributed as in $\mathsf{H}_{i,j}$.
- Finally, $\mathcal{A}_{\mathsf{OT}}$ passes $\mathbf{View}$ to D and outputs whatever D outputs.

In the above strategy, $\mathcal{A}_{\mathsf{OT}}$ has the same distinguishing advantage as D, which is non-negligible. However, since the protocol Π_{OT} is secure against a PPT semi-honest adversary corrupting the receiver, no such $\mathcal{A}_{\mathsf{OT}}$ can exist, and it follows that no such D can exist. So the distributions $\mathsf{H}_{i,j}$ and $\mathsf{H}_{i,j-1}$ are computationally indistinguishable.

Since none of the listed set of adjacent hybrids are distinguishable, it follows that the real view of the protocol $\mathsf{H}_{n,(n-d)}$ and the simulated view H_0 are computationally indistinguishable.

A.2 Proof of Theorem 4.1

Proof. The proof of this theorem follows in two parts. First, we show that there exists a PPT simulator $\mathsf{Sim}_{\mathsf{C}}$ that for any subset of parties $Z \in \mathcal{Z}_{\mathsf{C}} = 2^{\mathcal{P}}$ can output a view that is computationally indistinguishable to the view in the real protocol. Next, we show that there exists a simulator $\mathsf{Sim}_{\mathsf{S}}$ that for any subset of parties $Z \in \mathcal{Z}_{\mathsf{S}}$ can output a view that is statistically close to the view in the real protocol.

In order to examine the security of Figure 6 in the plain model, it becomes necessary to describe the properties of its building blocks:

1. Π_{stat} is a virtual n-party protocol computing the function f on input $\vec{x}$. Π_{stat} is secure in the presence of a semi-honest unbounded adversary corrupting any subset $Z \subset \mathcal{V}$ of the parties where $Z \in \mathcal{Z}_{\mathsf{S}}$. This means that there exists a simulator $\mathsf{Sim}_{\mathsf{stat}}$ [BGW88] that, given the inputs and outputs of the parties in Z can generate a view of this adversary that is statistically close to its view in the real execution of Π_{stat}.

$$\hat{\mathbf{View}}^Z_{\mathsf{stat}} \leftarrow \mathsf{Sim}_{\mathsf{stat}}(1^s, \{x_i\}_{V_i \in Z}, f(\vec{x}))$$

$$\left\{\hat{\mathbf{View}}^Z_{\mathsf{stat}}\right\}_{\kappa\in\mathbb{N},\vec{r}\in\mathcal{R}^n} \overset{s}{\equiv} \left\{\mathbf{View}^Z_{\mathsf{stat}}\right\}_{\kappa\in\mathbb{N},\vec{r}\in\mathcal{R}^n}$$

2. Much like how Π_{stat} can be written as the set of functions $\{\mathsf{NxtMsg}_i^j\}_{i\in[n],j\in[r]}$, the simulator $\mathsf{Sim}_{\mathsf{stat}}$ can also be written as,

$$\mathsf{Sim}_{\mathsf{stat}} = \{\mathsf{NxtMsg}_i^{j*}\}_{i\in[n],j\in[r]}$$

Here, for each corrupted party $V_i \in Z$, the functions $\{\mathsf{NxtMsg}_i^{j*}\}_{j\in[r]} = \{\mathsf{NxtMsg}_i^j\}_{j\in[r]}$ are as in the real protocol Π_{stat}. However, for each honest party $V_i \notin Z$, for each round $j \in [r]$, the next-message function has the same structure as the real function but produces simulated messages that do not depend on the party's input:

$$\{m_{i\to k}^{j'}\}_{k\neq i\in[n]} \leftarrow \mathsf{NxtMsg}_i^{j*}(\hat{\mathbf{View}}_{\mathsf{stat},i}^{j-1})$$

Here $\hat{\mathbf{View}}_{\mathsf{stat},i}^{j-1}$ is the simulated view with all the messages that the corrupt party sends to V_i up to round $j-1$. The set $\{m_{i\to k}^{j'}\}_{k\neq i\in[n]}$ is the set of simulated messages output on behalf of V_i to the other parties in round j.

3. The real protocol Π_{in} is composed of the set of protocols $\{\Pi_{\mathsf{NxtMsg}_i^j}\}_{i\in[n],j\in[r]}$ corresponding to the Π_{stat} in question. Each sub-protocol $\Pi_{\mathsf{NxtMsg}_i^j}$ works as described in Figure 4 with P_i as the evaluator and the rest of the parties as the garblers. Due to Theorem 3.4, we know that for a PPT semi-honest adversary corrupting $Z \subseteq \mathcal{P}$ there exists a simulator $\mathsf{Sim}_{\mathsf{C}}^{\mathsf{NxtMsg}_i^j}$ that given the inputs and outputs of the parties in Z can produce a view that is computationally indistinguishable to the distribution from a real execution of the sub-protocol:

$$\hat{\mathbf{View}}_{\mathsf{NxtMsg}_i^j}^{Z} \leftarrow \mathsf{Sim}_{\mathsf{C}}^{\mathsf{NxtMsg}_i^j}(1^\kappa, \{[\mathbf{View}_{\Pi_{\mathsf{stat}},i}^{j-1}]_k, M_k\}_{P_k\in Z}, \\ \{c_k = M_k \oplus \{[m_{i\to i'}^j]_k\}_{i'\neq i\in[n]}\}_{k\in[n]})$$

$$\{\hat{\mathbf{View}}_{\mathsf{NxtMsg}_i^j}^{Z}\}_{\kappa\in\mathbb{N},\vec{r}\in\mathcal{R}^n} \overset{\mathrm{c}}{\equiv} \{\mathbf{View}_{\mathsf{NxtMsg}_i^j}^{Z}\}_{\kappa\in\mathbb{N},\vec{r}\in\mathcal{R}^n}$$

We also have that for an unbounded semi-honest adversary corrupting $Z \subseteq \mathcal{P} - \{P_i\}$ there exists a simulator $\mathsf{Sim}_{\mathsf{S}}^{\mathsf{NxtMsg}_i^j}$ that, given the inputs and outputs of the parties in Z, can produce a view that is statistically close to the distribution from a real execution of the sub-protocol:

$$\hat{\mathbf{View}}_{\mathsf{NxtMsg}_i^j}^{Z} \leftarrow \mathsf{Sim}_{\mathsf{S}}^{\mathsf{NxtMsg}_i^j}(1^s, \{[\mathbf{View}_{\Pi_{\mathsf{stat}},i}^{j-1}]_k, M_k\}_{P_k\in Z}, \\ \{c_k = M_k \oplus \{[m_{i\to i'}^j]_k\}_{i'\neq i\in[n]}\}_{k\in[n]})$$

$$\{\hat{\mathbf{View}}_{\mathsf{NxtMsg}_i^j}^{Z}\}_{\kappa\in\mathbb{N},\vec{r}\in\mathcal{R}^n} \overset{\mathrm{s}}{\equiv} \{\mathbf{View}_{\mathsf{NxtMsg}_i^j}^{Z}\}_{\kappa\in\mathbb{N},\vec{r}\in\mathcal{R}^n}$$

Similarly to the above, if this simulator is given the inputs of the adversarial parties and the outputs are based on $\{[m_{i\to i'}^{j'}]_k\}_{i'\neq i\in[n]}\}_{k\in[n]}$, as output by the simulator of the virtual protocol $\mathsf{NxtMsg}_i^{j*} \in \mathsf{Sim}_{\mathsf{stat}}$, the view simulated is a view that is still statistically close to that in the real execution of $\Pi_{\mathsf{NxtMsg}_i^j}$:

$$\hat{\mathbf{View}}^Z_{\mathsf{NxtMsg}_i^{j*}} \leftarrow \mathsf{Sim}_{\mathsf{S}}^{\mathsf{NxtMsg}_i^j}(1^s, \{[\hat{\mathbf{View}}^{j-1}_{\mathsf{stat},i}]_k, M_k\}_{P_k \in Z},$$
$$\{c_k = M_k \oplus \{[m^{j'}_{i \to i'}]_k\}_{i' \neq i \in [n]}\}_{k \in [n]})$$

$$\{\hat{\mathbf{View}}^Z_{\mathsf{NxtMsg}_i^{j*}}\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n} \overset{\text{s}}{\equiv} \{\mathbf{View}^Z_{\mathsf{NxtMsg}_i^j}\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$$

Given these building-blocks, we can prove the theorem in the plain model.

Let us first consider the case of security against a PPT semi-honest adversary that corrupts any subset $Z \subseteq \mathcal{P}$ of the parties. The simulator for Figure 6, making black-box calls to simulators in $\{\mathsf{Sim}_{\mathsf{C}}^{\mathsf{NxtMsg}_i^j}\}_{i \in [n], j \in [r]}$, works as follows:

1. $\mathsf{Sim}_{\mathsf{C}}$ has the input of all the corrupt parties $\{x_i\}_{P_i \in Z}$ and the function output $f(\vec{x})$. It samples randomness $\{r_i\}_{P_i \in Z}$ for all corrupt parties.
2. In the '**Initialize shared state**' phase, $\mathsf{Sim}_{\mathsf{C}}$, on behalf of each honest party $P_{i'} \notin Z$, samples a random input $x_{i'} \leftarrow \{0,1\}$ and creates shares $\{[x_{i'}]_i\}_{i \in [n]} \leftarrow \mathsf{Share}_{n,n}(x_{i'})$. For each $i' \in [n]$, it creates shares also of the randomnesses $\{[r_{i'}]_i\}_{i \in [n]} \leftarrow \mathsf{Share}_{n,n}(r_{i'})$. It sends the set of shares $\{[x_{i'}]_i, [r_{i'}]_i\}_{P_{i'} \notin Z}$ to the corrupt party $P_i \in Z$.
3. For each virtual party $V_i \in \mathcal{V}$ and each round $j \in [r]$ of the virtual protocol Π_{stat}, for the protocol $\Pi_{\mathsf{NxtMsg}_i^j}$ as in Figure 4, $\mathsf{Sim}_{\mathsf{C}}$ makes a black-box call to the PPT simulator $\mathsf{Sim}_{\mathsf{C}}^{\mathsf{NxtMsg}_i^j}$ that can simulate the view of the corrupt parties in a way that is computationally indistinguishable from the real view. Let this set of views be,
$$\{\hat{\mathbf{View}}^Z_{\mathsf{NxtMsg}_i^j}\}_{i \in [n], j \in [r]}$$
4. In the phase to '**Derive the output**', $\mathsf{Sim}_{\mathsf{C}}$ gives the adversary shares of the output such that they reconstruct to $\{f(\vec{x}) = f_i(\vec{x})\}_{P_i \in Z}$

This completes the simulation. The view of the adversary in the above execution of $\mathsf{Sim}_{\mathsf{C}}$ is distributed as,

$$\left\{\{x_i\}_{P_i \in Z}, f(\vec{x}), \{r_i\}_{P_i \in Z}, \{\hat{\mathbf{View}}^Z_{\mathsf{NxtMsg}_i^j}\}_{i \in [n], j \in [r]}\right\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$$

It remains to argue that the above view is computationally indistinguishable to the real view of the adversary in Figure 6. Note that this view differs from the real view in the protocol in that the views of the next-message protocols are composed of calls to $\mathsf{Sim}_{\mathsf{C}}^{\mathsf{NxtMsg}_i^j}$ in the simulation. In order to show that the output of $\mathsf{Sim}_{\mathsf{C}}$ is indistinguishable from the real view, consider the following set of $rn + 1$ hybrids:

– Hybrid H_0. This is the distribution of the output of Sim_C.

$$\mathsf{H}_0 = \left\{ \{x_i\}_{P_i \in Z}, f(\vec{x}), \{r_i\}_{P_i \in Z}, \{\hat{\mathbf{View}}^Z_{\mathsf{NxtMsg}^j_i}\}_{i\in[n], j\in[r]} \right\}_{\kappa\in\mathbb{N}, \vec{r}\in\mathcal{R}^n}$$

– Hybrid $\mathsf{H}_{i,j}$. For each $i \in [n]$ and $j \in [r]$, this hybrid experiment contains the view of parties in Z in the real protocol execution up to the execution of the sub-protocol $\Pi_{\mathsf{NxtMsg}^j_i}$. The rest of the view is generated as in the simulation Sim_C.

$$\begin{aligned}\mathsf{H}_{i,j} = \Big\{ & \{x_{i'}\}_{P_{i'} \in Z}, f(\vec{x}), \{r_{i'}\}_{P_{i'} \in Z}, \\ & \{\mathbf{View}^Z_{\mathsf{NxtMsg}^{j'}_{i'}}\}_{i'<i,(i'=i,j'\le j)}, \\ & \{\hat{\mathbf{View}}^Z_{\mathsf{NxtMsg}^{j'}_{i'}}\}_{i'>i,(i'=i,j'>j)} \Big\}_{\kappa\in\mathbb{N}, \vec{r}\in\mathcal{R}^n}\end{aligned}$$

– Hybrid $\mathsf{H}_{n,r}$. Note that the last hybrid of the above form is distributed exactly as the real execution of Protocol 6.

$$\mathsf{H}_{n,r} = \left\{ \{x_i\}_{P_i \in Z}, f(\vec{x}), \{r_i\}_{P_i \in Z}, \{\mathbf{View}^Z_{\mathsf{NxtMsg}^j_i}\}_{i\in[n], j\in[r]} \right\}_{\kappa\in\mathbb{N}, \vec{r}\in\mathcal{R}^n}$$

Claim. Assuming that the protocol $\Pi_{\mathsf{NxtMsg}^j_i}$ is secure in the presence of a PPT semi-honest adversary corrupting any subset $Z \subseteq \mathcal{P}$ of the parties, the view in the hybrid distribution $\mathsf{H}_{i,j}$ is computationally indistinguishable from that in $\mathsf{H}_{i,j-1}$.

Proof. Note that in the hybrid distributions described above, each pair of adjacent hybrids $\mathsf{H}_{i,j}$ and $\mathsf{H}_{i,j-1}$ (similarly, H_0 and $\mathsf{H}_{1,1}$; and each pair $\mathsf{H}_{i,r}$ and $\mathsf{H}_{i+1,1}$) differ only the view of the execution of the sub-protocol $\Pi_{\mathsf{NxtMsg}^j_i}$. In $\mathsf{H}_{i,j-1}$, this view is the simulated view $\hat{\mathbf{View}}^Z_{\mathsf{NxtMsg}^j_i}$ output by $\mathsf{Sim}_\mathsf{C}^{\mathsf{NxtMsg}^j_i}$. In $\mathsf{H}_{i,j}$ this is the view of the parties in Z in the real execution of $\Pi_{\mathsf{NxtMsg}^j_i}$.

We show that if there existed a PPT distinguisher D that can distinguish between the hybrid distributions $\mathsf{H}_{i,j}$ and $\mathsf{H}_{i,j-1}$ with non-negligible advantage ϵ, then D can be used in a black-box way by a PPT adversary $\mathcal{A}$ that distinguishes between the distributions $\{\hat{\mathbf{View}}^Z_{\mathsf{NxtMsg}^j_i}\}_{\kappa\in\mathbb{N}, \vec{r}\in\mathcal{R}^n}$ from the output of $\mathsf{Sim}_\mathsf{C}^{\mathsf{NxtMsg}^j_i}$ and $\{\mathbf{View}^Z_{\mathsf{NxtMsg}^j_i}\}_{\kappa\in\mathbb{N}, \vec{r}\in\mathcal{R}^n}$ from the real execution of the sub-protocol as in Figure 4. The adversary $\mathcal{A}$ works as follows:

– $\mathcal{A}$ has the indices (i, j), the set of corrupted parties $Z \subseteq \mathcal{P}$ and $(\vec{x}, f)$. It samples randomness $\vec{r} \leftarrow \mathcal{R}^n$ for all parties.
– For the next message functions $\mathsf{NxtMsg}^{j'}_{i'}$ of Π_{stat} where $i' < i$ or $i' = i$ and $j' < j$, it creates the view as in the real protocol: $\mathbf{View}^Z_{\mathsf{NxtMsg}^{j'}_{i'}}$. For the instances where $i' > i$ or $i' = i$ and $j' > j$, it creates the view as in the simulation $\hat{\mathbf{View}}^Z_{\mathsf{NxtMsg}^{j'}_{i'}}$.

- Corresponding to the $(i,j)^{th}$ instance, $\mathcal{A}$ sends all the inputs and outputs to the challenger $\mathcal{C}$ that samples a bit b. If $b=0$, $\mathcal{C}$ computes,

$$\begin{aligned}\mathbf{View}_{\mathsf{NxtMsg}} \leftarrow \mathsf{Sim}_{\mathsf{C}}^{\mathsf{NxtMsg}_i^j}(1^\kappa, &\{[\mathbf{View}^{j-1}_{\Pi_{\mathsf{stat}},i}]_k, M_k\}_{P_k \in Z},\\ &\{c_k = M_k \oplus \{[m^j_{i\to i'}]_k\}_{i' \neq i \in [n]}\}_{k\in[n]})\end{aligned}$$

  Otherwise it computes $\mathbf{View}_{\mathsf{NxtMsg}}$ to be the views of the parties in Z in the real execution of $\Pi_{\mathsf{NxtMsg}_i^j}$.
- $\mathcal{C}$ gives $\mathbf{View}_{\mathsf{NxtMsg}}$ to $\mathcal{A}$ and, with this, $\mathcal{A}$ completes the view $\mathbf{View}$ to be input to D. Note that if $b=0$, $\mathbf{View}$ is distributed as in $\mathsf{H}_{i,j-1}$ and if $b=1$, it is distributed as in $\mathsf{H}_{i,j}$.
- Finally, $\mathcal{A}$ passes $\mathbf{View}$ to D and outputs whatever D outputs.

In the above strategy, $\mathcal{A}$ has the same distinguishing advantage as D, which is non-negligible. However, since the protocol $\Pi_{\mathsf{NxtMsg}_i^j}$ is secure in the presence of a PPT semi-honest adversary corrupting any subset $Z \subseteq \mathcal{P}$ of the parties, no such $\mathcal{A}$ can exist and therefore no such D can exist. So the distributions $\mathsf{H}_{i,j}$ and $\mathsf{H}_{i,j-1}$ are computationally indistinguishable.

Since none of the listed set of adjacent hybrids are distinguishable, it follows that the real view of the protocol $\mathsf{H}_{n,r}$ and the simulated view H_0 are computationally indistinguishable. Therefore, it follows that the view produced by $\mathsf{Sim}_{\mathsf{C}}$ is overall computationally indistinguishable to that of the real execution of Figure 6 for any $Z \in \mathcal{Z}_{\mathsf{C}}$.

Let us now consider the case of statistical security against an unbounded semi-honest adversary corrupting any $Z \in \mathcal{Z}_{\mathsf{S}}$. Let $Z_V \subset \mathcal{V}$ be the set of corrupted parties such that $Z_V = \{V_i\}_{P_i \in Z}$. The virtual protocol Π_{stat} is secure against an unbounded semi-honest adversary corrupting Z_V since $Z_V \in \mathcal{Z}_{\mathsf{S}}$. The simulator $\mathsf{Sim}_{\mathsf{S}}$ for protocol 6, making black-box calls to the simulators in $\{\mathsf{Sim}_{\mathsf{S}}^{\mathsf{NxtMsg}_i^j}\}_{i\in[n],j\in[r]}$, operates as follows:

1. $\mathsf{Sim}_{\mathsf{S}}$ has the input of all the corrupt parties $\{x_i\}_{P_i \in Z}$ and the function output $f(\vec{x})$. It samples randomness $\{r_i\}_{P_i\in Z}$ for all corrupt parties.
2. In the '**Initialize shared state**' phase, $\mathsf{Sim}_{\mathsf{S}}$, on behalf of each honest party $P_{i'} \notin Z$, samples a random input $x_{i'} \leftarrow \{0,1\}$ and creates shares $\{[x_{i'}]_i\}_{i\in[n]} \leftarrow \mathsf{Share}_{n,n}(x_{i'})$. For each $i' \in [n]$, it creates shares of the randomnesses $\{[r_{i'}]_i\}_{i\in[n]} \leftarrow \mathsf{Share}_{n,n}(r_{i'})$. It sends the set $\{[x_{i'}]_i, [r_{i'}]_i\}_{P_{i'}\notin Z}$ to the corrupt party $P_i \in Z$.
3. For each round $j \in [r]$, for each virtual party $V_i \in \mathcal{V}$ where $P_i \notin Z$ is honest, $\mathsf{Sim}_{\mathsf{S}}$ computes the inputs and outputs of the simulator $\mathsf{Sim}_{\mathsf{stat}}$ of the virtual protocol,

$$\{m^{j'}_{i\to k}\}_{k\neq i\in[n]} \leftarrow \mathsf{NxtMsg}_i^{j*}(\hat{\mathbf{View}}^{j-1}_{\mathsf{stat},i})$$

Then, shares of these inputs and outputs are used to generate the view in the real protocol:

$$\hat{\mathbf{View}}^{Z}_{\mathsf{NxtMsg}^{j*}_{i}} \leftarrow \mathsf{Sim}_{\mathsf{S}}^{\mathsf{NxtMsg}^{j}_{i}}(1^s, \{[\hat{\mathbf{View}}^{j-1}_{\mathsf{stat},i}]_k, M_k\}_{P_k \in Z}, \{c_k = M_k \oplus \{[m^{j'}_{i \to i'}]_k\}_{i' \neq i \in [n]}\}_{k \in [n]})$$

For each round $j \in [r]$, for each virtual party $V_i \in \mathcal{V}$ where $P_i \in Z$ is corrupted, $\mathsf{Sim}_{\mathsf{S}}$ computes the inputs and outputs of the real-world execution of the virtual protocol,

$$\{m^{j}_{i \to k}\}_{k \neq i \in [n]} \leftarrow \mathsf{NxtMsg}^{j}_{i}(\hat{\mathbf{View}}^{j-1}_{\mathsf{stat},i})$$

Then, shares of these inputs and outputs are used to generate the view in the real protocol:

$$\mathbf{View}^{Z}_{\mathsf{NxtMsg}^{j}_{i}} \leftarrow \Pi_{\mathsf{NxtMsg}^{j}_{i}}(1^s, \{[\hat{\mathbf{View}}^{j-1}_{\mathsf{stat},i}]_k, M_k\}_{k \in [n]})$$

In the end of this phase, the adversary's view consists of,

$$\{\hat{\mathbf{View}}^{Z}_{\mathsf{NxtMsg}^{j*}_{i}}\}_{j \in [r], P_i \notin Z}, \{\mathbf{View}^{Z}_{\mathsf{NxtMsg}^{j}_{i}}\}_{j \in [r], P_i \in Z}$$

4. In the phase to '**Derive the output**', $\mathsf{Sim}_{\mathsf{S}}$ gives the adversary shares of the output such that when combined with the shares that it derives from its own views, they reconstruct to $\{f(\vec{x}) = f_i(\vec{x})\}_{P_i \in Z}$

This completes the simulation. The view of the adversary in the above execution of $\mathsf{Sim}_{\mathsf{S}}$ is distributed as,

$$\left\{\{x_i\}_{P_i \in Z}, f(\vec{x}), \{r_i\}_{P_i \in Z}, \{\hat{\mathbf{View}}^{Z}_{\mathsf{NxtMsg}^{j*}_{i}}\}_{j \in [r], P_i \notin Z}, \{\mathbf{View}^{Z}_{\mathsf{NxtMsg}^{j}_{i}}\}_{j \in [r], P_i \in Z}\right\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$$

It remains to argue that the above view is statistically close to the real view of the adversary in Figure 6. For this, consider the following hybrids:

– Hybrid H_0. This is the distribution of the output of $\mathsf{Sim}_{\mathsf{S}}$.

$$\mathsf{H}_0 = \left\{\{x_i\}_{P_i \in Z}, f(\vec{x}), \{r_i\}_{P_i \in Z}, \{\hat{\mathbf{View}}^{Z}_{\mathsf{NxtMsg}^{j*}_{i}}\}_{j \in [r], P_i \notin Z}, \{\mathbf{View}^{Z}_{\mathsf{NxtMsg}^{j}_{i}}\}_{j \in [r], P_i \in Z}\right\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$$

– Hybrid H_1. An intermediate hybrid distribution with the simulated virtual protocol $\mathsf{Sim}_{\mathsf{stat}}$ for the view of Z_V but with each protocol execution among the real parties $\mathcal{P}$ being a real execution:

$$\mathsf{H}_1 = \Big\{ \{x_i\}_{P_i \in Z}, f(\vec{x}), \{r_i\}_{P_i \in Z}, \{\mathbf{View}^Z_{\mathsf{NxtMsg}_i^{j*}}\}_{j \in [r], P_i \notin Z}, \{\mathbf{View}^Z_{\mathsf{NxtMsg}_i^{j}}\}_{j \in [r], P_i \in Z} \Big\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$$

This differs from H_0 in that the view of real protocol corresponding to the next-message functions of the simulator of the virtual protocol is generated as in a real execution of Protocol 4, instead of using the simulation.

– Hybrid H_2. The distribution of the view of the parties in Z in a real execution of Protocol 6.

$$\mathsf{H}_2 = \Big\{ \{x_i\}_{P_i \in Z}, f(\vec{x}), \{r_i\}_{P_i \in Z}, \{\mathbf{View}^Z_{\mathsf{NxtMsg}_i^{j}}\}_{j \in [r], i \in [n]} \Big\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$$

Claim. Assuming that for each $i \in [n]$ and $j \in [r]$, the protocol $\mathit{\Pi}_{\mathsf{NxtMsg}_i^j}$ as in Protocol 4 is secure in the presence of a semi-honest unbounded adversary that can corrupt any subset of $\mathcal{P} - \{P_i\}$, the view in the hybrid distributions H_0 and H_1 are statistically close.

Proof. Let $(n-d)$ be the number of honest parties and let r be the number of rounds in $\mathit{\Pi}_{\mathsf{stat}}$. In order to show that the hybrids H_0 and H_1 are statistically close, consider the following set of $r(n-d)+1$ hybrids:

– Hybrid A_0. This is a hybrid that is distributed identically to H_0.

$$A_0 = \Big\{ \{x_i\}_{P_i \in Z}, f(\vec{x}), \{r_i\}_{P_i \in Z}, \{\hat{\mathbf{View}}^Z_{\mathsf{NxtMsg}_i^{j*}}\}_{j \in [r], P_i \notin Z}, \{\mathbf{View}^Z_{\mathsf{NxtMsg}_i^{j}}\}_{j \in [r], P_i \in Z} \Big\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$$

– Hybrid $A_{i,j}$. For each $i \in [n-d]$ and $j \in [r]$, this hybrid distribution is derived using the next message functions of the simulated virtual protocol $\mathsf{Sim}_{\mathsf{stat}}$ where for the sub-protocols up to $\mathit{\Pi}_{\mathsf{NxtMsg}_i^{j*}}$ a view of the parties in Z of the real execution of the protocol is included, instead of a simulated one as in H_0. The rest of the view is generated as in $\mathsf{Sim}_{\mathsf{S}}$.

$$\begin{aligned} A_{i,j} = \Big\{ & \{x_{i'}\}_{P_{i'} \in Z}, f(\vec{x}), \{r_{i'}\}_{P_{i'} \in Z}, \\ & \{\mathbf{View}^Z_{\mathsf{NxtMsg}_{i'}^{j'*}}\}_{i' < i, (i'=i, j' \le j)}, \\ & \{\hat{\mathbf{View}}^Z_{\mathsf{NxtMsg}_{i'}^{j'*}}\}_{i' > i, (i'=i, j' > j)}, \\ & \{\mathbf{View}^Z_{\mathsf{NxtMsg}_{i'}^{j'}}\}_{j' \in [r], P_{i'} \in Z} \Big\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n} \end{aligned}$$

– Hybrid $A_{n-d,r}$. Note that the last hybrid distribution as above is distributed the same way as in H_1.

$$A_{n-d,r} = \Big\{ \{x_i\}_{P_i \in Z}, f(\vec{x}), \{r_i\}_{P_i \in Z}, \{\mathbf{View}^Z_{\mathsf{NxtMsg}_i^{j*}}\}_{j \in [r], P_i \notin Z}, \{\mathbf{View}^Z_{\mathsf{NxtMsg}_i^{j}}\}_{j \in [r], P_i \in Z} \Big\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$$

In the above set of hybrids, note that adjacent pairs of hybrids $A_{i,j}$ and $A_{i,j-1}$ (similarly, A_0 and $A_{1,1}$ and each $A_{i,r}$ and $A_{i+1,1}$) differ only in the view of one execution of the sub-protocol $\Pi_{\mathsf{NxtMsg}_i^{j*}}$ where $\mathsf{NxtMsg}_i^{j*} \in \mathsf{Sim}_{\mathsf{stat}}$. In $A_{i,j-1}$, the view of this execution is the simulated view $\hat{\mathbf{View}}^Z_{\mathsf{NxtMsg}_i^{j*}}$. In $A_{i,j}$, this is $\mathbf{View}^Z_{\mathsf{NxtMsg}_i^{j*}}$, as in the real execution. The rest of the hybrid is distributed the same way. We already have that the simulated view produced by $\mathsf{Sim}_{\mathsf{S}}^{\mathsf{NxtMsg}_i^j}$ is distributed in a way that is statistically close to the real view of the protocol $\Pi_{\mathsf{NxtMsg}_i^{j*}}$. Let ϵ be the statistical difference between the distributions $\{\hat{\mathbf{View}}^Z_{\mathsf{NxtMsg}_i^{j*}}\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$ and $\{\mathbf{View}^Z_{\mathsf{NxtMsg}_i^{j*}}\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$ of the simulated and real executions respectively. Then the statistical difference between the hybrids is no more than ϵ. It also follows from the triangle inequality of statistical differences that the difference between the simulated view H_0 output by $\mathsf{Sim}_{\mathsf{S}}$ and the hybrid H_1 is $\leq r(n-d)\epsilon$. Therefore, these hybrid distributions are statistically close.

Claim. Assuming that the protocol Π_{stat} is secure in the presence of a semi-honest unbounded adversary with adversary structure $\mathcal{Z}_{\mathsf{S}}$, the view in the hybrid distributions H_1 and H_2 are statistically close.

Proof. We already have that the distribution of the view produced by $\mathsf{Sim}_{\mathsf{stat}}$ is statistically close to that in the real execution of the protocol Π_{stat}. Let ϵ be the statistical difference between these views. Since this is the only difference between the two hybrids H_1 and H_2, the statistical difference between them is also no more than ϵ. Therefore, these hybrid distributions are statistically close.

So it follows that the view produced by $\mathsf{Sim}_{\mathsf{S}}$ is statistically close to that of the real execution of Figure 6 for any $Z \in \mathcal{Z}_{\mathsf{S}}$.

A.3 Proof of Lemma 5.6

Proof. The proof for Lemma 5.6 follows in two parts. First we show that Protocol 5.4 is secure in the presence of a PPT malicious adversary corrupting any subset of the parties $Z \subseteq \mathcal{P}$. Next, we show that the same protocol is secure in the presence of an unbounded malicious adversary corrupting a subset of parties $Z \in \mathcal{Z}_{\mathsf{S}}$. In order to examine the security of Protocol 5.4 in the plain model, it becomes necessary to first prove the following lemma:

Lemma A.1 *Let* $\mathsf{Com}_{\mathsf{SB}}$ *be a statistically binding commitment. Let* $\mathsf{Com}_{\mathsf{SH}}$ *be a statistically hiding commitment and let* SZKAoK *be a statistical zero-knowledge argument of knowledge protocol. Let* ZKPoK *be a zero-knowledge proof of knowledge protocol. Then the* n*-party commitment protocol* Π_{Com} *in Figure 12 is a maliciously fall-back secure realization of* $\mathcal{F}_{\mathsf{Com}}$ *tolerating* $\mathcal{Z}_{\mathsf{S}}$*.*

Proof. The proof for this lemma follows in 2 parts. First, we need to show that Figure 12 securely realizes $\mathcal{F}_{\mathsf{Com}}$ in the presence of a PPT malicious adversary corrupting any subset of the parties $Z \subseteq \mathcal{P}$. Next we need to show that the same protocol realizes the functionality in the presence of an unbounded malicious adversary corrupting a subset of parties $Z \in \mathcal{Z}_{\mathsf{S}}$.

Security against a PPT Adversary. Let $\mathcal{A}$ be a PPT adversary corrupting any subset of parties $Z \subset \mathcal{P}$. There exists a PPT simulator $\mathsf{Sim}_{\mathsf{C}}$ that simulates the view of $\mathcal{A}$ in the protocol in the ideal world, in a way that is computationally indistinguishable from the view in the real execution. $\mathsf{Sim}_{\mathsf{C}}$ would operate as follows:

- If C is honest,
  1. In the **commit phase** of Π_{Com}, first for each $P_i \in Z$ participate honestly in the commitment protocol Com with P_i as the committer. Note that for each such corrupted committer, this sub-protocol is computationally binding:
     - A PPT adversary corrupting an arbitrary subset of parties in $\mathcal{P}$ including the committer cannot equivocate the view of an opening to the commitment to the honest parties. This holds due to the fact that the statistically binding commitment used here for the input shares is also computationally binding. Further, the protocol also uses a statistically hiding commitment to commitment to the input as a whole, to each viewer. This scheme is computationally binding and so a PPT adversary cannot equivocate, no matter how large the collusion set.

     Then for each $P_i \notin Z$, such that $P_i \neq C$, it acts honestly as in the protocol and creates the commitments with Com. Note that for each such honest committer, this sub-protocol is computationally hiding:
     - A PPT adversary corrupting an arbitrary subset of parties in $\mathcal{P}$ excluding the committer cannot know any information about the value committed to. This holds directly from the fact that a statistically hiding commitment is also computationally hiding and the statistically binding commitment is computationally hiding as well. So there must exist a simulator $\mathsf{Sim}_{\mathsf{C}}^{\mathsf{Com}}$ that can simulate the view of the adversary in a way that is computationally indistinguishable from the real view.
  2. For each $j \in [\kappa]$ it samples x^* at random and honestly computes what the committer C would with input x^*.
  3. All parties decommit to Com from step 1 and $\mathsf{Sim}_{\mathsf{C}}$ learns r.

4. Now $\mathsf{Sim_C}$ rewinds to Step 2 and for each $j \in [\kappa]$, it first samples $\nu_j \leftarrow \{0, 1\}$ at random and for the matrix M_j both elements in the $r[j]^{th}$ column is set as ν_j. In the other column, at random, one element is set as ν_j and the other is set as $\nu_j + 1$.
5. Then the rest of the commit phase is executed honestly. Note here that the protocol ECom is also computationally hiding:
   - This is true due to the same reasons as for Com. Additionally, the SZKAoK and ZKPoK protocols used are zero-knowledge in the presence of PPT verifiers.
6. In the **decommit phase** of Π_{Com}, simulator $\mathsf{Sim_C}$ learns x from $\mathcal{F}_{\mathsf{Com}}$.
7. Now, for each $j \in [\kappa]$, in the $(\neg r[j])^{th}$ column of matrix M_j, the simulator de-commits to the element containing $\nu_j + x[j]$.

– If $\mathcal{A}$ corrupts C,
1. In the **commit phase** of Π_{Com}, first for each $P_i \in Z$ participate honestly in the commitment protocol Com with P_i as the committer. Then for each $P_i \notin Z$, such that $P_i \neq C$, it acts honestly as in the protocol and creates the commitments with Com.
2. For each $j \in [\kappa]$ it engages with the adversary $\mathcal{A}$ corrupting C in the 4 executions of ECom where,
   - Along with being computationally binding, the value committed in ECom is extractable. That is within the 'commit phase', when a PPT adversary is corrupting an arbitrary subset of parties in $\mathcal{P}$ including the committer, there exists an extractor that can interact with the adversary in the ideal execution and extract the input of the committer. Such an extractor $\mathsf{ECom}.E^*_{\mathsf{C}}$ would make a black-box call to the extractor of the SZKAoK, deriving the input to the statistically hiding commitment.

   So the simulator extracts from ECom all the elements of $\{M_j\}_{j\in[\kappa]}$ that C commits to.
3. These are used to derive x and this is given to $\mathcal{F}_{\mathsf{Com}}$. If this extraction fails, or the matrices are not well-formed, send $\perp$ to $\mathcal{F}_{\mathsf{Com}}$.
4. The rest of the protocol is executed honestly by the simulator on behalf of the honest parties.

Let $\mathbf{View}^{\mathsf{Com}}_{Z,i}(x)$ be the view of the adversary in a real execution of the protocol Com where party P_i is the committer with input x. Similarly, let $\mathbf{View}^{\mathsf{ECom}}_{Z,i}(x)$ be the view of the adversary in a real execution of the protocol ECom where party P_i is the committer with input x. For the case that the committer C is honest, the view produced by $\mathsf{Sim_C}$ is distributed as:

$$\Big\{ \{\mathbf{View}^{\mathsf{Com}}_{Z,i}(r_i)\}_{i\in[n]}, r, \{\mathbf{View}^{\mathsf{ECom}}_{Z,i}(M^*_j[a,b])\}_{j\in[\kappa],a,b\in\{0,1\}},$$
$$\{M^*_j[r[j],a]\}_{j\in[\kappa],a\in\{0,1\}}, \{M^*_j[\neg r[j], b_j]\}_{j\in[\kappa],b_j:M^*_j[\neg r[j],b_j]-M^*_j[r[j],a]=x} \Big\}$$

This differs from that in the real distribution only in Step 4 where for all $j \in [\kappa]$, in one column, $M^*_j[\neg r[j], \cdot]$, of matrix M^*_j, the two values committed to using ECom are not equal. This is not the case in the real distribution where both values in the column not checked in M_j are equal and they equal the sum of the value opened and the committer's input to the commitment protocol. This amounts to having κ different commitments ECom in $\mathsf{Sim_C}$ with different messages from that in the real view. Everything other than these commitments are created identically. In order to argue that the view produced by the real protocol is distributed computationally indistinguishable from in the simulation, consider the following set of $\kappa + 1$ hybrids:

- H_0 : Let this be the distribution of the view as produced in the simulation.
- $\forall j \in [\kappa], \mathsf{H}_j$: Let this be the distribution of the view in which the first j matrices committed to $M_1, \cdots, M_j$ are created as in the real distribution and the rest are created as in the simulation.
  The last such hybrid H_κ is distributed as in the real view.

Note here that adjacent hybrids $\forall j \in [\kappa], \mathsf{H}_j$ and H_{j-1} differ only in the value of one element in the matrix M_j. A PPT distinguisher that can distinguish between these views with non-negligible advantage can be used in a black-box way by a PPT adversary $\mathcal{A}$ to break the computational hiding property of ECom. This is done by $\mathcal{A}$ first constructing the complete view except M_j, then the three other elements of M_j (the ones opened in the cut-and-choose, and the one opened in the release phase) and committing to them. The last element of M_j is directly replaced by the challenge commitment. If the underlying message equals the value opened in the release phase, the hybrid is H_j. Otherwise, this forms hybrid H_{j-1}. The adversary $\mathcal{A}$ passes this to the distinguisher and outputs whatever it outputs. Such an adversary would have the same advantage as the distinguisher, which is non-negligible. But since ECom is computationally hiding and no such adversary can exist, no such distinguisher can exist. Hence the distributions of the simulated and real views are computationally indistinguishable.

For the case that the committer C is corrupted, the view produced by $\mathsf{Sim_C}$ as indicated above, differs from that in the real distribution only in Step 2 where for all $j \in [\kappa]$, the extractor for ZKPoK is executed for all ECom executions for each element in the matrix M_j. This is not the case in the real distribution where the ECom protocol is honestly executed. This amounts to having 4κ different commitment executions ECom in $\mathsf{Sim_C}$. However, the transcripts produced by these are identical to that in the real execution. Therefore, the real and simulated views are identically distributed. This holds for all executions for which a cheating committer is caught cheating in the cut-and-choose. If, however, a committer cheats in the cut-and-choose execution and is not caught in the real execution, its view would differ from the simulation where cheating will always be detected. The probability of success corresponds to the event in which it can guess the complete challenge r correctly and this happens with probability $\frac{1}{2^\kappa}$ which is negligible in κ.

Security against an Unbounded Adversary. Let $\mathcal{A}$ be an unbounded adversary corrupting any subset of parties $Z \in \mathcal{Z}_\mathsf{S}$. There exists a PPT simulator

Sim_S that simulates the view of $\mathcal{A}$ in the protocol in the ideal world, in a way that is statistically indistinguishable from the view in the real execution. Sim_S would operate as follows:

– If C is honest,
  1. In the **commit phase** of Π_{Com}, first for each $P_i \in Z$ participate honestly in the commitment protocol Com with P_i as the committer. Note that for each such corrupted committer, this sub-protocol is statistically binding:
     - An unbounded adversary corrupting any $Z \in \mathcal{Z}_\mathsf{S}$ that includes the committer cannot equivocate the view of an opening to that commitment to the honest parties. This is ensured owing to the fact that the committer uses a 2-party statistically binding commitment to interact with all other parties. It shares its input using $\mathsf{SS}_{\mathcal{Z}_\mathsf{S}}$ and so it follows that the shares of the honest parties, when reconstructed uniquely determine the input. Therefore, corrupted viewers cannot change this value by falsely claiming to have shares.

     Then for each $P_i \notin Z$, such that $P_i \neq C$, it acts honestly as in the protocol and creates the commitments with Com. Note that for each such honest committer, this sub-protocol is statistically hiding:
     - An unbounded adversary corrupting any $Z \in \mathcal{Z}_\mathsf{S}$ not including the committer cannot know any information about the value committed to. This holds since the 2-party commitment used to commit to the input is statistically hiding. While it may be true that the statistically binding commitment is not hiding for this adversary, this is only used to commit to shares of the input. As such, since $\mathsf{SS}_{\mathcal{Z}_\mathsf{S}}$ is used for secret-sharing, the adversary corrupting Z will never have enough shares to reconstruct the secret. So there exists a simulator $\mathsf{Sim}_\mathsf{S}^{\mathsf{Com}}$ that can simulate the view of the adversary in a way that is statistically close to the real view.
  2. For each $j \in [\kappa]$ it samples x^* at random and honestly computes what the committer C would with input x^*.
  3. All parties decommit to Com from step 1 and Sim_S learns r.
  4. Now Sim_S rewinds to Step 2 and for each $j \in [\kappa]$, it first samples $\nu_j \leftarrow \{0,1\}$ at random and for the matrix M_j both elements in the $r[j]^{th}$ column is set as ν_j. In the other column, at random, one element is set as ν_j and the other is set as $\nu_j + 1$.
  5. Then the rest of the commit phase is executed honestly. Note here that the protocol ECom is also statistically hiding:
     - This also holds the same way as in Com. Additionally, although the ZKPoK used is only zero-knowledge for a PPT verifier, an unbounded adversarial verifier getting the witnesses still only gets shares of a $\mathsf{SS}_{\mathcal{Z}_\mathsf{S}}$ secret-sharing and cannot reconstruct the input.
  6. In the **decommit phase** of Π_{Com}, simulator Sim_S learns x from $\mathcal{F}_{\mathsf{Com}}$.
  7. Now, for each $j \in [\kappa]$, in the $(\neg r[j])^{th}$ column of matrix M_j, the simulator de-commits to the element containing $\nu_j + x[j]$.

– If $\mathcal{A}$ corrupts C,

1. In the **commit phase** of Π_{Com}, first for each $P_i \in Z$ participate honestly in the commitment protocol Com with P_i as the committer. Then for each $P_i \notin Z$, such that $P_i \neq C$, it acts honestly as in the protocol and creates the commitments with Com.
2. For each $j \in [\kappa]$ it engages with the adversary $\mathcal{A}$ corrupting C in the 4 executions of ECom where,
   - ECom is statistically binding for Z, same way as Com is. Additionally, note that the SZKAoK used is only sound for a PPT prover, but an unbounded adversarial prover still cannot cheat as it can't equivocate in the statistically binding commitment.
   - Within the 'commit phase' of ECom, when an unbounded adversary corrupts any $Z \in \mathcal{Z}_{\mathsf{S}}$ including the committer, there exists an extractor that can interact with the adversary in the ideal execution and extract the input of the committer. For this case, such an extractor $\Pi_{\mathsf{ECom}}.E^*_{\mathsf{S}}$ would work by participating in the commit phase on behalf of all the honest parties and making black-box calls to the extractor of the ZKPoK, deriving the shares that were committed to in each statistically binding commitment. Once all the honest parties' shares have been extracted, these can be reconstructed to get the committer's input.

   So the simulator extracts from ECom all the elements of $\{M_j\}_{j\in[\kappa]}$ that C commits to.
3. These are used to derive x and this is given to $\mathcal{F}_{\mathsf{Com}}$. If this extraction fails, or the matrices are not well-formed, send $\perp$ to $\mathcal{F}_{\mathsf{Com}}$.
4. The rest of the protocol is executed honestly by the simulator on behalf of the honest parties.

Let $\mathbf{View}^{\mathsf{Com}}_{Z,i}(x)$ be the view of the adversary in a real execution of the protocol Com where party P_i is the committer with input x. Similarly, let $\mathbf{View}^{\mathsf{ECom}}_{Z,i}(x)$ be the view of the adversary in a real execution of the protocol ECom where party P_i is the committer with input x. For the case that the committer C is honest, the view produced by $\mathsf{Sim}_{\mathsf{S}}$ is distributed as:

$$\Big\{ \{\mathbf{View}^{\mathsf{Com}}_{Z,i}(r_i)\}_{i\in[n]}, r, \{\mathbf{View}^{\mathsf{ECom}}_{Z,i}(M^*_j[a,b])\}_{j\in[\kappa],a,b\in\{0,1\}},$$
$$\{M^*_j[r[j],a]\}_{j\in[\kappa],a\in\{0,1\}}, \{M^*_j[\neg r[j],b_j]\}_{j\in[\kappa],b_j:M^*_j[\neg r[j],b_j]-M^*_j[r[j],a]=x} \Big\}$$

This differs from that in the real distribution only in Step 4 where for all $j \in [\kappa]$, in one column of matrix M^*_j, the two values committed to using ECom are not equal. This is not the case in the real distribution where both values in the column not checked are equal and they equal the sum of the value opened and the value committed to. This amounts to having κ different commitments ECom in $\mathsf{Sim}_{\mathsf{S}}$ with different messages from that in the real view. In order to argue that the view produced by the real protocol is distributed statistically indistinguishable than in the simulation, consider the following set of $\kappa + 1$ hybrids:

- H_0 : Let this be the distribution of the view as produced in the simulation.
- $\forall j \in [\kappa], \mathsf{H}_j$: Let this be the distribution of the view in which the first j matrices committed to $M_1, \cdots, M_j$ are created as in the real distribution and the rest are created as in the simulation.
  The last such hybrid H_κ is distributed as in the real view.

Note here that adjacent hybrids $\forall j \in [\kappa], \mathsf{H}_j$ and H_{j-1} differ only in the value of one element in the matrix M_j and M_j^* respectively. Such hybrid distributions are statistically close since ECom is a statistically hiding commitment. Since there are κ many such hybrids, it follows that the real and simulated distribution are statistically close.

For the case that the committer C is corrupted, the view produced by Sim_S differs from that in the real distribution only in Step 2 where for all $j \in [\kappa]$, the extractor for SZKAoK is executed for all ECom executions for each element in the matrix M_j. This is not the case in the real distribution where the ECom protocol is honestly executed. This amounts to having 4κ different commitment executions ECom in Sim_S. However, the transcripts produced by these are identical to that in the real execution. Therefore, the real and simulated views are identically distributed.

It therefore follows that Figure 12 realizes $\mathcal{F}_{\mathsf{Com}}$ with fall-back security in the presence of malicious adversaries.

Proof of Security for the Offline Phase against a PPT Adversary. We start by showing that for any PPT malicious adversary $\mathcal{A}$ corrupting any subset of parties $Z \subseteq \mathcal{P}$, there exists a PPT simulator Sim_C that can interact with it for an ideal execution of $\mathcal{F}_{\mathsf{AuthTriples}}$ (Figure 7) and produce a view that is computationally indistinguishable from the real view of Protocol 5.4. The simulator Sim_C works by playing the adversary in the execution of $\Pi_{\mathsf{AuthTriples}}$ and simulates the execution of the compiled protocol to $\mathcal{A}$. Sim_C works as follows:

1. **Simulating the Coin Toss Phase.** In both the executions of the commit phase of Π_{coin}, Sim_C participates honestly with the following exception:
   - Sim_C uses $\Pi_{\mathsf{Com}}.\mathsf{Sim}_\mathsf{C}$ in each commitment protocol to extract the inputs of the corrupt parties. If this succeeds, compute $\vec{r}$ for the **watch-list** or $\vec{d}$ for the **degree test** in the **Check Phase** as required.
   - If extraction fails and returns $\perp$, then Sim_C sets $\vec{r} = \perp$ or $\vec{d} = \perp$ as required and continues the execution.
2. **Simulating the Input Commitment Phase.** For all $i \in [n]$, Sim_C participates in the commit phase of each execution Π^i_{Com} as follows:
   - If P_i is honest, then Sim_C samples the elements in $\vec{v_i}$ at random and honestly computes the protocol on its behalf, giving $\{\mathbf{View}^{i,k}_{\mathsf{Com},j,v}\}_{j \in [m], P_k \in Z}$ to $\mathcal{A}$. Here, each $\mathbf{View}^{i,k}_{\mathsf{Com},j,v}$ contains commitments that P_i gives to P_k corresponding to the j^{th} virtual server execution.

- If $P_i \in Z$, Sim_C receives $\{\mathbf{View}^{i,k}_{\mathsf{Com},j,v}\}_{j\in[m],P_k\in\mathcal{P}-Z}$. For each $j \in [m]$, makes a black-box call to the extractor $\Pi_{\mathsf{ECom}}.\mathsf{Sim}_\mathsf{C}$ and collects $\{\vec{v_i^j}\}_{j\in[m]}$.
- If this extractor fails and returns $\perp$, then Sim_C stores $\vec{v_i^j} = \perp$ as the extracted input for this execution of Π^i_{Com}.

At the end of this phase, Sim_C has the set of inputs $\{\vec{v_i^j}\}_{j\in[m],i\in[n]}$ out of which the adversary's inputs are extracted and that of the honest parties is generated by Sim_C. Note again that this set may contain multiple $\perp$ symbols corresponding to executions in which extraction had failed.

3. **Input Extraction.**
   - Sim_C computes for each $P_i \in Z$, the inputs $\vec{v_i} \leftarrow \mathsf{Recon}_{t,m}(\{\vec{v_i^j}\}_{j\in[m]})$ where reconstruction is done with error-correction.
   - If reconstruction succeeds for all $P_i \in Z$, Sim_C gives $\{\vec{v_i}\}_{P_i\in Z}$ to $\mathcal{F}_{\mathsf{AuthTriples}}$ and receives the tuples $\{\vec{s_i}\}_{P_i\in Z}$ as output. It sets $\mathsf{Flag} = \mathsf{TRUE}$.
   - If for any $P_i \in Z$, the reconstruction fails, then Sim_C sends $\perp$ to $\mathcal{F}_{\mathsf{AuthTriples}}$ and does not receive an output. It internally sets $\mathsf{Flag} = \mathsf{FALSE}$.
     For each $P_i \in Z$, it samples $r^i \leftarrow \mathbb{F}$ uniformly at random and assigns it to the output vector $\vec{s_i}$. All other elements of this vector are as in the (partially reconstructed) input vector $\vec{v_i}$.
     If $P_1 \in Z$, it samples $c_1^{j'}, \mathsf{MAC}_1(a^{j'}), \mathsf{MAC}_1(b^{j'}), \mathsf{MAC}_1(c^{j'}) \leftarrow \mathbb{F}$ also uniformly at random and assigns it to $\vec{s_1}$. The remaining elements of this vector are as in the (partially reconstructed) input vector $\vec{v_1}$.
4. **Setting the Inputs and Outputs of each Π^j_S.** This is a step in which Sim_C performs some internal computation, without interacting with the adversary $\mathcal{A}$, in preparation for the **Compute Phase**.
   - If $\mathsf{Flag} = \mathsf{FALSE}$ and the interaction with $\mathcal{F}_{\mathsf{AuthTriples}}$ was not successful, then on behalf of each honest party $P_i \in \mathcal{P} - Z$, Sim_C sets $\vec{v_i^j}$ as sampled and committed to in the **Input Commitment Phase** as the input to the execution Π^j_S. The randomness used is $r_{i,j}$ as will be derived in the **Randomness Generation Phase** next.
   - If $\mathsf{Flag} = \mathsf{TRUE}$ but $\vec{r} = \perp$, then also the same actions as above are taken.
   - Otherwise if $\mathsf{Flag} = \mathsf{TRUE}$ and $\vec{r} \neq \perp$, then Sim_C knows which executions of Π^j_S will be checked in the **watch-list** step and which will not.

   The details of the simulators actions for the last case are given below:
   - For each watched instance $j \in \vec{r}$, Sim_C internally simulates the actions of the adversary assuming it is semi-honest (not by interacting with the malicious adversary $\mathcal{A}$), and derives the outputs and randomness of the malicious parties.
     - Sim_C sets $\{r_{i,j}\}_{i\in[n]}$ sampled uniformly at random as the randomness used by each party in this execution.
     - It sets $\{\vec{v_i^j}\}_{i\in[n]}$ as derived from the **Input Commitment Phase** as the input used by each party in this execution.
     - Sim_C internally computes Π^j_S using this input and randomness to get $\{\vec{s_i^j}\}_{i\in[n]}$. Let $\mathbf{View}_{\Pi^j_S}$ be the view generated in this computation.

The outputs of these executions and the output of $\mathcal{F}_{\mathsf{AuthTriples}}$ are required to set the outputs of the corrupt parties in the un-watched instances, that need to be input to $\mathsf{Sim}_{\mathsf{C}}^{\Pi_S^j}$.

– For each un-watched instance $j \in [m] - \vec{r}$, it suffices to initially only compute the output of the corrupted parties.
  - For each $P_i \in Z$, $\mathsf{Sim}_{\mathsf{C}}$ has $r^i \in \vec{s_i}$ from $\mathcal{F}_{\mathsf{AuthTriples}}$, or otherwise, sampled at random. It computes $\{[r^i]_j\}_{j\in[m]} \leftarrow \mathsf{Share}_{t,m}(r^i)$ under the constraint that $\{[r^i]_j\}_{j\in\vec{r}}$ are exactly as in the view of the watched executions $\{\mathbf{View}_{\Pi_S^j}\}_{j\in\vec{r}}$ computed above.
  - For each $P_i \in Z$ such that $i \neq 1$, set the input as $\vec{s_i^j}$ as extracted from the **Input Commitment Phase**. Set the output as,

$$\vec{s_i^j} = \begin{cases} [\Delta_i]_j \in \vec{v_i^j}, [r^i]_j & \\ [r_i^k]_j, [\mathsf{MAC}_i(r^k)]_j \in \vec{v_i^j} & \forall k \in [n] \\ [a_i^{j'}]_j, [b_i^{j'}]_j, [c_i^{j'}]_j \in \vec{v_i^j} & \forall j' \in [\mathsf{T}] \\ [\mathsf{MAC}_i(a^{j'})]_j, [\mathsf{MAC}_i(b^{j'})]_j, [\mathsf{MAC}_i(c^{j'})]_j \in \vec{v_i^j} & \forall j' \in [\mathsf{T}] \end{cases}$$

  - If party P_1 is corrupted, then $\forall k \in [n]$, $\mathsf{Sim}_{\mathsf{C}}$ has received $\mathsf{MAC}_1(r^k) \in \vec{s_1}$. It computes $\{[\mathsf{MAC}_1(r^k)]_j\}_{j\in[m]} \leftarrow \mathsf{Share}_{2t,m}(\mathsf{MAC}_1(r^k))$ such that $\{[\mathsf{MAC}_1(r^k)]_j\}_{j\in\vec{r}}$ are exactly as in the watched execution views $\{\mathbf{View}_{\Pi_S^j}\}_{j\in\vec{r}}$.
    Similarly, $\forall j' \in [\mathsf{T}]$, $\mathsf{Sim}_{\mathsf{C}}$ has received the computed values for $c_1^{j'}, \mathsf{MAC}_1(a^{j'}), \mathsf{MAC}_1(b^{j'}), \mathsf{MAC}_1(c^{j'}) \in \vec{s_1}$. It computes the sharings,

$$\begin{aligned} \{[c_1^{j'}]_j\}_{j\in[m]} &\leftarrow \mathsf{Share}_{2t,m}(c_1^{j'}) \\ \{[\mathsf{MAC}_1(a^{j'})]_j\}_{j\in[m]} &\leftarrow \mathsf{Share}_{2t,m}(\mathsf{MAC}_1(a^{j'})) \\ \{[\mathsf{MAC}_1(b^{j'})]_j\}_{j\in[m]} &\leftarrow \mathsf{Share}_{2t,m}(\mathsf{MAC}_1(b^{j'})) \\ \{[\mathsf{MAC}_1(c^{j'})]_j\}_{j\in[m]} &\leftarrow \mathsf{Share}_{4t,m}(\mathsf{MAC}_1(c^{j'})) \end{aligned}$$

    such that the shares corresponding to $j \in \vec{r}$ are exactly as in the watched execution views $\{\mathbf{View}_{\Pi_S^j}\}_{j\in\vec{r}}$. Then the output is set as,

$$\vec{s_1^j} = \begin{cases} [\Delta_1]_j \in \vec{v_1^j}, [r^1]_j & \\ [r_1^k]_j \in \vec{v_1^j}, [\mathsf{MAC}_1(r^k)]_j & \forall k \in [n] \\ [a_1^{j'}]_j, [b_1^{j'}]_j \in \vec{v_1^j} & \forall j' \in [\mathsf{T}] \\ [c_1^{j'}]_j, [\mathsf{MAC}_1(c^{j'})]_j, [\mathsf{MAC}_1(a^{j'})]_j, [\mathsf{MAC}_1(b^{j'})]_j & \forall j' \in [\mathsf{T}] \end{cases}$$

  - Sim_C internally executes $\mathsf{Sim}_{\mathsf{C}}^{\Pi_S^j}$ with these inputs and outputs and generates the random tapes $\{r_{i,j}\}_{P_i\in Z}$ of the corrupt parties.

5. **Simulating the Randomness Generation Phase.** For all $i \in [n], j \in [m]$, $\mathsf{Sim}_{\mathsf{C}}$ participates in the commit phase of each execution $\Pi_{\mathsf{CTW}}^{i,j}$ as follows:

- For each honest party $P_k \in \mathcal{P} - Z$ such that $k \neq i$, Sim_C executes the simulator for **commit phase** $\Pi_\mathsf{Com}.\mathsf{Sim}_\mathsf{C}$.
- For each corrupt party $P_k \in Z$ such that $k \neq i$, Sim_C receives from $\mathcal{A}$ the set of views $\{\mathbf{View}_{\mathsf{Com}}^{k,k',i,j}\}_{P_{k'} \in \mathcal{P}-Z}$. Each such view contains the commitments that P_k gives to $P_{k'}$.
- It also makes a black-box call to the extractor $\Pi_\mathsf{Com}.\mathsf{Sim}_\mathsf{C}$ and receives $r_{i,j}^k$. If this extractor fails and returns $\perp$, then Sim_C stores $r_{i,j} = \perp$ as the extracted randomness for this execution of $\Pi_\mathsf{CTW}^{i,j}$.
- Then Sim_C sets the input in $\Pi_\mathsf{Com}.\mathsf{Sim}_\mathsf{C}$ for each honest party such that they sum to $r_{i,j}$ as given in Step 4. This execution is completed until the end of the commit phase.
- For each $P_k \in \mathcal{P}$ such that $k \neq i$, Sim_C performs the **decommit phase** of each $\Pi_\mathsf{Com}.\mathsf{Sim}_\mathsf{C}$ and receives from $\mathcal{A}$ the randomness $\{r_{i,j}^k\}_{P_k \in Z}$. If the adversary fails to decommit properly, Sim_C sends ABORT on behalf of the honest parties and halts the execution.
- If extraction had succeeded for the execution of $\Pi_\mathsf{ECom}.\mathsf{Sim}_\mathsf{C}$, Sim_C computes $r_{i,j} = \sum_{k \in [n]} r_{i,j}^k$.

At the end of this phase, Sim_C has the set of randomness $\{r_{i,j}\}_{i \in [n], j \in [m]}$ where the adversary's input randomnesses are as decided in Step 4. Note that this set may contain multiple $\perp$ symbols corresponding to executions in which extraction had failed.

6. **Simulating the Compute Phase.** If $\mathsf{Flag} = \mathsf{FALSE}$, or $\mathsf{Flag} = \mathsf{TRUE}$ but $\vec{r} = \perp$, then for each $j \in [m]$, Sim_C participates honestly in an execution of Π_S^j as in the real execution, on behalf of each honest party. It runs each honest party $P_i \in \mathcal{P} - Z$ using the input $\vec{v_i^j}$ and randomness $r_{i,j}$.
Otherwise, if $\mathsf{Flag} = \mathsf{TRUE}$ and $\vec{r} \neq \perp$, then for each $j \in [m]$, Sim_C simulates the watched and un-watched instances Π_S^j differently.
   - For each watched instance $j \in \vec{r}$, Sim_C uses the honest party's generated input shares $\{\vec{v_i^j}\}_{P_i \in \mathcal{P}-Z}$ and randomness $\{r_{i,j}\}_{P_i \in \mathcal{P}-Z}$ and interacts with the adversary $\mathcal{A}$ as in a real execution of the protocol Π_S^j. Let $\mathbf{View}^*_{\Pi_S^j}$ be the view of the adversary in this interaction.
   - For each un-watched instance $j \in [m] - \vec{r}$,
     - Sim_C internally uses the PPT semi-honest simulator $\mathsf{Sim}_\mathsf{C}^{\Pi_S^j}$ with the adversary $\mathcal{A}$'s inputs as $\{\vec{v_i^j}\}_{P_i \in Z}$, the randomness $\{r_{i,j}\}_{P_i \in Z}$ and output as $\{\vec{s_i^j}\}_{P_i \in Z}$. Let this view be $\mathbf{View}_{\Pi_S^j}$.
     - Note that Sim_C, in its interaction with $\mathcal{A}$, can detect where $\mathcal{A}$ deviates from semi-honest behaviour as it can compare the messages in the interaction with the expected messages in the view generated in $\mathbf{View}_{\Pi_S^j}$.
     - If there is an inconsistency and $\mathcal{A}$ has behaved maliciously, this corresponds to that execution Π_S^j being corrupted. Since the protocol $\Pi_\mathsf{AuthTriples}$ is secure in the presence of semi-honest adaptive corruption, this corresponds to Sim_C adaptively corrupting all the honest

parties $\mathcal{P} - Z$ in the execution of $\Pi_{\mathsf{AuthTriples}}$. $\mathsf{Sim_C}$ uses $\{\vec{s_i^j}, \vec{v_i^j}\}_{P_i \in Z}$ to create the inputs of the honest parties as follows:

- Let $P_{i^*} \in \mathcal{P} - Z$ be a designated honest party. For all honest parties $P_i \neq P_{i^*}$, sample the values in $\vec{v_i^j}$ uniformly at random.
- It remains to determine the inputs for P_{i^*}. This is set as follows:
  - · For each $P_k \in Z$, get $[r^k]_j \in \vec{s_k^j}$, and the set $\{[r_i^k]_j \in \vec{v_i^j}\}_{P_i \neq P_{i^*} \in \mathcal{P}}$. Set each $[r_{i^*}^k]_j = [r^k]_j - \sum_{i \neq i^* \in [n]} [r_i^k]_j$ and assign $[r_{i^*}^k]_j$ to $\vec{v_{i^*}^j}$.
  - · If $P_1 \notin Z$, then sample the rest of the values in $\vec{v_{i^*}^j}$ uniformly at random.
  - · Otherwise, sample $[\Delta_{i^*}]_j \leftarrow \mathbb{F}$ and for all $j' \in [\mathsf{T}]$, $[a_{i^*}^{j'}]_j, [b_{i^*}^{j'}]_j \leftarrow \mathbb{F}$ uniformly at random and assign them to $\vec{v_{i^*}^j}$.
    For each $k \in [n]$, set $[\mathsf{MAC}_{i^*}(r^k)]_j$ such that computation in functionality $\mathcal{F}_S$ (Figure 8) for $[\mathsf{MAC}'(r^k)]_j$ is the difference between $[\mathsf{MAC}_1(r^k)]_j \in \vec{v_1^j}$ and $[\mathsf{MAC}_1(r^k)]_j \in \vec{s_1^j}$.
    Similarly, for each $j' \in [\mathsf{T}]$, set the shares $([c_{i^*}^{j'}]_j, [\mathsf{MAC}_{i^*}(a^{j'})]_j, [\mathsf{MAC}_{i^*}(b^{j'})]_j, [\mathsf{MAC}_{i^*}(c^{j'})]_j)$ such that the computation in $\mathcal{F}_S$ (Figure 8) gives the difference in the output shares in $\vec{s_1^j}$ and input shares in $\vec{v_1^j}$. These are also assigned in $\vec{v_{i^*}^j}$.

$\mathsf{Sim_C}$ sends the inputs of the honest parties $\{\vec{v_i^j}\}_{P_i \in \mathcal{P} - Z}$ to $\mathsf{Sim_C}^{\Pi_S^j}$.

- $\mathsf{Sim_C}^{\Pi_S^j}$ gives to $\mathsf{Sim_C}$ the randomness that explains the transcript thus far with respect to these inputs. Then $\mathsf{Sim_C}$ uses this to continue the the execution of Π_S^j by interacting with $\mathcal{A}$.

7. **Simulating the Check Phase.** Here, $\mathsf{Sim_C}$ first honestly executes the protocol by emulating the honest parties in the interaction with $\mathcal{A}$. If any of the checks fail or $\vec{r} = \perp$, the simulator sends ABORT to the adversary.
   If $\mathsf{Flag} = \mathsf{FALSE}$ then also $\mathsf{Sim_C}$ sends ABORT to the adversary. $\mathsf{Sim_C}$ additionally checks if among all $j \in [m] - \vec{r}$, there exist more than t instances of Π_S^j in which $\mathcal{A}$ has deviated from the protocol. If this holds, $\mathsf{Sim_C}$ sends ABORT in the interaction with the adversary.
   Otherwise $\mathsf{Sim_C}$ accepts.

Note that for every adversary, if the execution of the protocol in the real world aborts, so does the simulated execution in the ideal world. Additionally, with negligible probability, it may be the case that the simulation aborts but the real execution does not. This happens when there is no ABORT in the check phase, but more that t un-watched executions of Π_S^j are corrupted. Another case where possibly the real execution can be accepting but the simulation aborts is if $\mathsf{Flag} = \mathsf{FALSE}$ in the simulation but there is no abort in the check phase. However, note that if $\mathsf{Flag} = \mathsf{FALSE}$, this would imply that the inputs of the corrupted parties could not be extracted and are therefore not well-formed. If this occurs the degree-test would fail except with negligible probability in the size of the field $\mathbb{F}$. If the execution doesn't abort, the view generated by $\mathsf{Sim_C}$ is distributed as,

$$\begin{Bmatrix}\{\mathbf{View}_{\Pi^{i,j}_{\mathsf{CTW}}}\}_{i\in[n],j\in[m]}, \{\mathbf{View}_{\Pi^{i}_{\mathsf{Com}}}\}_{i\in[n]}, \{\mathbf{View}_{\Pi^{\mathcal{R}_t}_{\mathsf{coin}}}, \mathbf{View}_{\Pi^{\mathbb{F}^{2n+6\mathsf{T}+3}}_{\mathsf{coin}}}\}, \\ \{\mathbf{View}^*_{\Pi^j_S} \leftarrow \Pi^j_S(\{\vec{v_i^j}\}_{i\in[n]}; \{r_{i,j}\}_{i\in[n]})\}_{j\in\vec{r}}, \\ \{\mathbf{View}_{\Pi^j_S} \leftarrow \mathsf{Sim}^{\Pi^j_S}_{\mathsf{C}}(\{\vec{v_i^j}\}_{P_i\in Z}; \{r_{i,j}\}_{P_i\in Z}; \{\vec{s_i^j}\}_{P_i\in Z})\}_{j\in[m]-\vec{r}}\end{Bmatrix}$$

It remains to argue that the above view is computationally indistinguishable from the real view in Protocol 5.4. For this, consider the following hybrids:

1. Hybrid H_0. This is constructed as the view of the environment in the real execution of the protocol.
2. Hybrid H_1. This is constructed in the same way as H_0, with the exception that in this experiment, the protocol aborts even in the case where there is no ABORT in the check phase, but more than t un-watched executions of Π^j_S are corrupted.
   The distribution of this view is statistically close to the that of the real distribution of hybrid H_0. This stems from the fact that, as stated above, the probability of the difference in the abort conditions is negligible in the security parameter κ.
3. Hybrid H_2. This is constructed in the same way as H_1, except for the view in each un-watched execution of Π^j_S. For each of these executions, the view is generated using a call to the PPT semi-honest adaptive simulator $\mathsf{Sim}^{\Pi^j_S}_{\mathsf{C}}$ with the correctly committed inputs and randomness, and outputs as would have been generated in H_1 for the corrupted parties.
   The distributions of H_1 and H_2 can be shown as computationally indistinguishable using a set of $m - \frac{t}{2} + 1$ intermediate hybrid distributions wherein each hybrid H'_j has real executions views for each un-watched execution up to the j^{th} run, and the rest of the views are simulated. Adjacent such hybrids differ only in one execution of an un-watched Π^j_S. Such hybrids can be shown as computationally indistinguishable by reducing to the fall-back security of protocol Π^j_S. Hence, it follows that H_1 and H_2 are also computationally indistinguishable.
4. Hybrid H_3. In this hybrid, in the 'Input Commitment Phase', on behalf of the honest parties, different inputs are used corresponding to all the un-watched executions as compared to the inputs to the real protocol. The rest of the distribution is created using these inputs of the honest parties, as in H_2.
   The distributions of H_2 and H_3 can be shown as computationally indistinguishable by reducing to the 'fall-back secure hiding property' of the n-party commitment Π_{Com}. Let s be the number of executions of such commitment protocols where an honest party is a committer. Then we can define $s + 1$ different hybrids where in each hybrid H'_k all the executions of Π_{Com} up to the k^{th} execution uses the real inputs of the honest parties in the protocol execution. Each execution beyond this generates its inputs independently

and as in the simulation $\mathsf{Sim}_{\mathsf{C}}$. Each pair of adjacent hybrids differ only in the input to one execution of Π_{Com} and such adjacent hybrids can be show as computationally indistinguishable by reducing to the fall-back secure hiding property of Π_{Com}.

5. Hybrid H_4. In this hybrid, in the 'Randomness Generation Phase', for all the parties, the commitments are created just as in $\mathsf{Sim}_{\mathsf{C}}^{\Pi_{\mathsf{Com}}}$. The rest of distribution is created as in H_3.
   The distributions of H_4 and H_3 can be shown as computationally indistinguishable by reducing to the indistinguishability of the real execution of the n-party commitment Π_{Com} and the output distribution of $\mathsf{Sim}_{\mathsf{C}}^{\Pi_{\mathsf{Com}}}$. Let s be the number of executions of such commitment protocols. Then we can define $s+1$ different hybrids where in each hybrid H'_k all the executions of Π_{Com} up to the k^{th} execution is the real execution of the commitment protocol. Each execution beyond this generates its inputs independently and as in the simulation $\mathsf{Sim}_{\mathsf{C}}$. Each pair of adjacent hybrids differ only in the input to one execution of Π_{Com} and such adjacent hybrids can be show as computationally indistinguishable by reducing to security of the simulation of Π_{Com}.
6. Hybrid H_5. This hybrid is distributed in exactly the same way as the the output of $\mathsf{Sim}_{\mathsf{C}}$. This conceptually differs from the distribution of H_4 in that the honest parties' inputs are never used in creating its contents and the randomness for all corrupt parties is generated by the simulator $\mathsf{Sim}_{\mathsf{C}}^{\Pi_S^j}$.
   In both hybrids, the watched executions are generated and distributed the same way. For the un-watched executions, the output of corrupt parties that is input to the semi-honest simulator $\mathsf{Sim}_{\mathsf{C}}^{\Pi_S^j}$ is generated using the output of the functionality $\mathcal{F}_{\mathsf{AuthTriples}}$ instead of being directly generated from the inputs of the honest party in the real protocol. However, both of these are identically distributed since in the real protocol, the complete protocol output would be generated in the same way as $\mathcal{F}_{\mathsf{AuthTriples}}$ does. For each individual execution of Π_S^j, the output is set as a share of a correct t-out-of-m secret-sharing in the simulation, same as in the real execution. Hence the hybrids H_5 and H_4 are identically distributed.

Claim. The view in the hybrid distributions H_0 and H_1 are statistically close.

Proof. The hybrid H_0 is distributed as in the view of the environment in the real execution of the protocol. The hybrid H_1 differs from this only in that for the protocol execution where it has not aborted in the check phase, but more than t un-watched semi-honest executions of Π_S^j are corrupted, the complete offline phase protocol execution aborts. Let X be the set of indices of corrupted Π_S^j executions and $\vec{r}$ be the set of $\frac{t}{2}$ checked instances. This case happens with the following probability over a random choice of $\vec{r}$:

$$\Pr_{\vec{r}\in\mathcal{R}_t}\left[(X\cap\vec{r}=\Phi)\wedge(|X|>t)\right]\leq\frac{\binom{m-t}{\frac{t}{2}}}{\binom{m}{\frac{t}{2}}}\leq\left(1-\frac{t}{m}\right)^{\frac{t}{2}}$$

This is negligible in the security parameter $\kappa = t$. It therefore follows that the two distributions are statistically close.

Claim. Assuming that the protocol Π_S^j is secure in the presence of a semi-honest PPT adaptive adversary arbitrarily corrupting the set of parties, the hybrid distributions H_1 and H_2 are computationally indistinguishable.

Proof. In order to show that the distributions of H_1 and H_2 are computationally indistinguishable consider the following set of $m - \frac{t}{2} + 1$ intermediate hybrid distributions:

- Hybrid $\mathsf{H}'_0 = \mathsf{H}_2$. This hybrid distribution contains simulated views of all the un-watched protocol executions of Π_S.
- Hybrid H'_j. For each $j \in [m - \frac{t}{2}]$, this hybrid experiment has real executions views for each un-watched execution of Π_S^j up to the j^{th} run, and the rest of the views are simulated.
- Hybrid $\mathsf{H}'_{m-\frac{t}{2}} = \mathsf{H}_1$. In this last hybrid, the views of all the un-watched executions of the protocols Π_S are real views of the semi-honest protocol execution.

Adjacent such hybrids above differ only in one execution of an un-watched Π_S^j. We show that if there existed a distinguisher D that can distinguish between the adjacent hybrid distributions H'_j and H'_{j-1} with non-negligible advantage ϵ, then D can be used in a black-box way by a PPT adversary $\mathcal{A}$ that distinguishes between the simulated distribution $\{\mathbf{View}_{\Pi_S^j}\}_{\kappa\in\mathbb{N},\vec{r}\in\mathcal{R}^n}$ output by $\mathsf{Sim}_{\mathsf{C}}^{\Pi_S^j}$ and the real distribution $\{\mathbf{View}^*_{\Pi_S^j}\}_{\kappa\in\mathbb{N},\vec{r}\in\mathcal{R}^n}$ of the protocol Π_S^j. The adversary $\mathcal{A}$ works as follows:

- $\mathcal{A}$ has the index j, the set of corrupt parties $Z \subseteq \mathcal{P}$ and the complete set of inputs. It samples randomness for all the parties.
- It begins generating the view of the offline protocol exactly as in the real execution until before the compute phase. In the compute phase, for all the watched instances of the semi-honest virtual protocol, it creates a real view of the protocol. For each un-watched instance up to the execution of Π_S^{j-1}, the view is generated according to the real execution again.
- For the execution of Π_S^j, give the input and randomness of all the parties to the challenger. It will interact with the adversary (on behalf of all the honest parties) and return a view $\mathbf{View}^j$ that is created either according to the real or the simulated distribution by the adaptive corruption simulator.
- The rest of the un-watched execution view are created as in the simulation of the virtual protocol. The view of the whole protocol is then completed as in the real distribution, except that the execution aborts each time it would have aborted in the simulation.
- This completed view is passed onto the distinguisher D and then $\mathcal{A}$ outputs whatever D outputs.

In the above strategy, $\mathcal{A}$ has the same distinguishing advantage as D, which is non-negligible. However, since the protocol Π_S^j is secure in the presence of a PPT semi-honest adaptive adversary corrupting any subset $Z \subseteq \mathcal{P}$ of the parties, no such $\mathcal{A}$ can exist and therefore no such D can exist. Hence, it follows that H_1 and H_2 are also computationally indistinguishable.

Claim. Assuming that the protocol Π_{Com} securely realizes $\mathcal{F}_{\mathsf{Com}}$ producing a computationally hiding view in the presence of a malicious PPT adversary arbitrarily corrupting any set of the parties, the hybrid distributions H_2 and H_3 are computationally indistinguishable.

Proof. Let s be the number of executions of Π_{Com} in the protocol where an honest party is the committer. The distributions of H_2 and H_3 can be shown as computationally indistinguishable by considering the following set of $s+1$ intermediate hybrid distributions:

– Hybrid $\mathsf{H}'_0 = \mathsf{H}_3$. This is the hybrid distribution where all the inputs used for the honest parties are as in the protocol simulation $\mathsf{Sim}_{\mathsf{C}}$, independent of the real protocol inputs. The view produced here is composed of the view of these commitment protocols, and that of the rest of the real offline protocol created depending on these inputs, with the exception, of course, that all the un-watched executions of the virtual protocol are replaced by simulated views, and the protocol aborts whenever the $\mathsf{Sim}_{\mathsf{C}}$ aborts.
– Hybrid H'_k. For each $k \in [s]$, in this hybrid, in up to the k^{th} execution of the commitment protocol, all the inputs used for the honest parties are as in the real execution and the rest are as in the protocol simulation $\mathsf{Sim}_{\mathsf{C}}$.
– Hybrid $\mathsf{H}'_s = \mathsf{H}_2$. This hybrid distribution contains all honest party inputs and commitment executions as in the real execution of the protocol. The rest of the view of the offline protocol is generated with this as the basis.

Note that for $k \in [s]$, each pair of adjacent hybrids H'_k and H'_{k-1} differ only in that the k^{th} execution of Π_{Com} uses a different input. The rest of the view is generated on its basis, in the same way. Let x be the input to this protocol in the real execution and x' be this input in the simulation. We show that if there existed a distinguisher D that can distinguish between the adjacent hybrid distributions H'_k and H'_{k-1} with non-negligible advantage ϵ, then D can be used in a black-box way by a PPT adversary $\mathcal{A}$ that distinguishes between the distribution $\{\mathbf{View}_{\Pi_{\mathsf{Com}}}(x)\}_{\kappa\in\mathbb{N},\vec{r}\in\mathcal{R}^n}$ as in $\mathsf{Sim}_{\mathsf{C}}$ and the real distribution $\{\mathbf{View}_{\Pi_{\mathsf{Com}}}(x')\}_{\kappa\in\mathbb{N},\vec{r}\in\mathcal{R}^n}$. The adversary $\mathcal{A}$ works as follows:

– $\mathcal{A}$ has the index k, the set of corrupt parties $Z \subseteq \mathcal{P}$ and the complete set of inputs including x and x'. It samples randomness for all the parties.
– It begins generating the view of the offline protocol exactly as in the real execution until before the input commitment phase. In this phase, for all instances of Π_{Com} where an honest party is the committer, up to the $k-1^{th}$ execution, the view is generated using the inputs as in the real execution.
– For the k^{th} execution of Π_{Com}, give the inputs x and x' to the challenger. It will return a view $\mathbf{View}^k$ that is created either using x as the input to the commitment or with x'.

- The rest of the commitment execution views are created as in the simulation $\mathsf{Sim}_{\mathsf{C}}$. The view of the whole protocol is then completed as in the real distribution, except that all the un-watched virtual protocol executions are replaced by their simulations and the execution aborts each time it would have aborted in the simulation.
- This completed view is passed onto the distinguisher D and then $\mathcal{A}$ outputs whatever D outputs.

In the above strategy, $\mathcal{A}$ has the same distinguishing advantage as D, which is non-negligible. However, since the protocol Π_{Com} is secure in the presence of a PPT malicious adversary corrupting any subset $Z \subseteq \mathcal{P}$ of the parties and will produce a view that computationally hides the input, no such $\mathcal{A}$ can exist and therefore no such D can exist. Hence, it follows that H_3 and H_2 are also computationally indistinguishable.

Claim. Assuming that the protocol Π_{Com} securely realizes $\mathcal{F}_{\mathsf{Com}}$ in the presence of a malicious PPT adversary arbitrarily corrupting any set of the parties, the hybrid distributions H_3 and H_4 are computationally indistinguishable.

Proof. Let s be the number of executions of Π_{Com} in the protocol in the randomness generation phase. The distributions of H_4 and H_3 can be shown as computationally indistinguishable by considering the following set of $s+1$ intermediate hybrid distributions:

- Hybrid $\mathsf{H}'_0 = \mathsf{H}_4$. This is the distribution in which all the commitment protocol executions in the randomness generation phase are replaced by executions of $\mathsf{Sim}_{\mathsf{C}}^{\Pi_{\mathsf{Com}}}$.
- Hybrid H'_k. For all $k \in [s]$, in this hybrid, in up to the k^{th} instance of the commitment protocol, the real protocol Π_{Com} is executed, and all other executions are replaced by executions of $\mathsf{Sim}_{\mathsf{C}}^{\Pi_{\mathsf{Com}}}$.
- Hybrid $\mathsf{H}'_s = \mathsf{H}_3$. In this distribution, all the commitment protocol executions in the randomness generation phase are executions of the real protocol Π_{Com}.

Note that for $k \in [s]$, each pair of adjacent hybrids H'_k and H'_{k-1} differ only in the view of the k^{th} execution of Π_{Com} in the randomness generation phase. We show that if there existed a distinguisher D that can distinguish between the adjacent hybrid distributions H'_k and H'_{k-1} with non-negligible advantage ϵ, then D can be used in a black-box way by a PPT adversary $\mathcal{A}$ that distinguishes between the distribution $\{\mathbf{View}_{\Pi_{\mathsf{Com}}}\}_{\kappa\in\mathbb{N},\vec{r}\in\mathcal{R}^n}$ as output by $\mathsf{Sim}_{\mathsf{C}}^{\Pi_{\mathsf{Com}}}$ and the real distribution $\{\mathbf{View}^*_{\Pi_{\mathsf{Com}}}\}_{\kappa\in\mathbb{N},\vec{r}\in\mathcal{R}^n}$ in the protocol Π_{Com}. The adversary $\mathcal{A}$ works as follows:

- $\mathcal{A}$ has the index k, the set of corrupt parties $Z \subseteq \mathcal{P}$ and the complete set of inputs. It samples randomness for all the parties.
- It begins generating the view of the offline protocol as in the simulation until before the randomness generation phase. In this phase, for all instances of Π_{Com} up to the $k-1^{th}$ execution, the view is generated as in the real execution.

- For the k^{th} execution of Π_{Com}, give the input and randomness to the challenger. It will return a view $\mathbf{View}^k$ that is created either according to the real protocol execution or the simulation $\mathsf{Sim}_{\mathsf{C}}^{\Pi_{\mathsf{Com}}}$.
- The rest of the commitment execution views are created as in the simulation $\mathsf{Sim}_{\mathsf{C}}$. The view of the whole protocol is then completed as in the real distribution, except that all the un-watched virtual protocol executions are replaced by their simulations and the execution aborts each time it would have aborted in the simulation.
- This completed view is passed onto the distinguisher D and then $\mathcal{A}$ outputs whatever D outputs.

In the above strategy, $\mathcal{A}$ has the same distinguishing advantage as D, which is non-negligible. However, since the protocol Π_{Com} is secure in the presence of a PPT malicious adversary corrupting any subset $Z \subseteq \mathcal{P}$, no such $\mathcal{A}$ can exist and therefore no such D can exist. Hence, it follows that H_3 and H_4 are also computationally indistinguishable.

Since we have now shown that hybrid $\mathsf{H}_0 \overset{c}{\equiv} \mathsf{H}_4$ and $\mathsf{H}_4 \equiv \mathsf{H}_5$, it holds that the simulated and the real view of the protocol are computationally indistinguishable.

Security for the Offline Phase against an Unbounded Adversary. Next we need to show that for any unbounded malicious adversary $\mathcal{A}$ corrupting any subset of parties $Z \in \mathcal{Z}_{\mathsf{S}}$, there exists a PPT simulator $\mathsf{Sim}_{\mathsf{S}}$ that can interact with it for an ideal execution of $\mathcal{F}_{\mathsf{AuthTriples}}$ (Figure 7) and produce a view that is statistically indistinguishable from the real view of Protocol 5.4. The simulator $\mathsf{Sim}_{\mathsf{S}}$ works in exactly the same way as $\mathsf{Sim}_{\mathsf{C}}$ described above, with the following exceptions:

- In the **Randomness Generation Phase**, **Commit Phase**, and **Coin Toss Phase**, for each instance of the ‘commit phase’ of the commitment protocol Π_{Com}, input extraction for the corrupt parties is no longer done using calls to $\Pi_{\mathsf{Com}}.\mathsf{Sim}_{\mathsf{C}}$. Instead, for each honest party $P_i \in \mathcal{P} - Z$, $\mathsf{Sim}_{\mathsf{S}}$ works by calling $\Pi_{\mathsf{Com}}.\mathsf{Sim}_{\mathsf{S}}$ to extract the inputs.
  If this extraction or reconstruction fails, then $\mathsf{Sim}_{\mathsf{S}}$ works the same way as $\mathsf{Sim}_{\mathsf{C}}$ would if a call to $\Pi_{\mathsf{Com}}.\mathsf{Sim}_{\mathsf{C}}$ returns $\perp$.
- **Simulating the Compute Phase.** When $\mathsf{Flag} = \mathsf{TRUE}$ and $\vec{r} \neq \perp$, then for each un-watched instance $j \in [m] - \vec{r}$, $\mathsf{Sim}_{\mathsf{S}}$ works by using the PPT semi-honest simulator $\mathsf{Sim}_{\mathsf{S}}^{\Pi_S^j}$ instead of $\mathsf{Sim}_{\mathsf{C}}^{\Pi_S^j}$. It is used in the same way as $\mathsf{Sim}_{\mathsf{C}}$ does in the above simulation.

If the execution does not abort, the view generated by $\mathsf{Sim}_{\mathsf{S}}$ is distributed as,

$$\Big\{\{\mathbf{View}_{\Pi^{i,j}_{\mathsf{CTW}}}\}_{i\in[n],j\in[m]},\{\mathbf{View}_{\Pi^{i}_{\mathsf{Com}}}\}_{i\in[n]},\{\mathbf{View}_{\Pi^{\mathcal{R}_t}_{\mathsf{coin}}},\mathbf{View}_{\Pi^{\mathbb{F}^{2n+6\mathsf{T}+3}}_{\mathsf{coin}}}\},$$
$$\{\mathbf{View}^*_{\Pi^j_S} \leftarrow \Pi^j_S(\{\vec{v^j_i}\}_{i\in[n]};\{r_{i,j}\}_{i\in[n]})\}_{j\in\vec{r}},$$
$$\{\mathbf{View}_{\Pi^j_S} \leftarrow \mathsf{Sim}_{\mathsf{S}}^{\Pi^j_S}(\{\vec{v^j_i}\}_{P_i\in Z};\{r_{i,j}\}_{P_i\in Z};\{\vec{s^j_i}\}_{P_i\in Z})\}_{j\in[m]-\vec{r}}\Big\}$$

It remains to argue that the above view is statistically indistinguishable from the real view in Protocol 5.4. For this, consider the following hybrids:

1. Hybrid H_0. This is constructed as the view of the environment in the real execution of the protocol.
2. Hybrid H_1. This is constructed the same way as H_0, except that in this experiment, the protocol aborts even in the case where the check phase does not ABORT, but more than t un-watched executions of Π^j_S are corrupted.
   The distribution of this view is statistically close to the that of the real distribution of hybrid H_0. This stems from the fact that the probability of the difference in the abort conditions is negligible in the security parameter κ. It has been formally shown in the security proof in the presence of a PPT adversary.
3. Hybrid H_2. This is constructed in the same way as H_1, except for the view in each un-watched execution of Π^j_S. For each of these executions, the view is generated using a call to the PPT semi-honest adaptive simulator $\mathsf{Sim}_{\mathsf{S}}^{\Pi^j_S}$ with the correctly committed inputs and randomness, and outputs as would have been generated in H_1 for the corrupted parties.
   The hybrid distributions H_1 and H_2 can be shown to be statistically indistinguishable owing to the fact that each $\mathsf{Sim}_{\mathsf{S}}^{\Pi^j_S}$ produces a view that is statistically close to the real view of the semi-honest protocol for any $Z \in \mathcal{Z}_{\mathsf{S}}$.
4. Hybrid H_3. In this hybrid, in the 'Input Commitment Phase', on behalf of the honest parties, different inputs are used corresponding to all the un-watched executions as compared to the inputs to the real protocol. The rest of distribution is created as in H_2.
   The distributions of H_2 and H_3 can be shown as statistically indistinguishable owing to the 'fall-back secure hiding property' of the n-party extractable commitment Π_{Com}.
5. Hybrid H_4. In this hybrid, in the 'Randomness Generation Phase', for all the parties, the commitments are created just as in $\mathsf{Sim}_{\mathsf{S}}^{\Pi_{\mathsf{Com}}}$. The rest of distribution is created as in H_3.
   The distributions of H_4 and H_3 can be shown as statistically indistinguishable owing to the security of the simulation $\mathsf{Sim}_{\mathsf{S}}^{\Pi_{\mathsf{Com}}}$ of the n-party extractable commitment Π_{Com}.
6. Hybrid H_5. This hybrid is distributed in exactly the same way as the the output of $\mathsf{Sim}_{\mathsf{S}}$. This conceptually differs from the distribution of H_4 in that the honest parties' inputs are never used in creating its contents and the

randomness for all corrupt parties is generated by $\mathsf{Sim}_{\mathsf{S}}^{\Pi_S^j}$.
However since the outputs of the corrupt parties derived from the ideal functionality and that in the real execution are identically distributed, it follows that both hybrids H_5 and H_4 are identically distributed.

Claim. Assuming that the protocol Π_S^j is secure in the presence of a semi-honest computationally unbounded adaptive adversary with adversary structure $\mathcal{Z}_{\mathsf{S}}$, the hybrid distributions H_1 and H_2 are statistically close.

Proof. In order to show that the distributions of H_1 and H_2 are statistically close consider the following set of $m - \frac{t}{2} + 1$ intermediate hybrid distributions:

– Hybrid $\mathsf{H}'_0 = \mathsf{H}_2$. This hybrid distribution contains simulated views of all the un-watched protocol executions of Π_S.
– Hybrid H'_j. For each $j \in [m - \frac{t}{2}]$, this hybrid experiment has real executions views for each un-watched execution of Π_S^j up to the j^{th} run, and the rest of the views are simulated.
– Hybrid $\mathsf{H}'_{m-\frac{t}{2}} = \mathsf{H}_1$. In this last hybrid, the views of all the un-watched executions of the protocols Π_S are real views of the semi-honest protocol execution.

Adjacent such hybrids above differ only in one execution of an un-watched Π_S^j. Let ϵ be the statistical difference between the view output by $\mathsf{Sim}_{\mathsf{S}}^{\Pi_S^j}$ and that in the real execution, which is negligible. Then the statistical difference between the adjacent hybrids can be no more than ϵ. It also follows from the triangle inequality of statistical differences that the difference between the hybrid distributions H_1 and H_2 is $\leq (m - \frac{t}{2})\epsilon$. Therefore these distributions are statistically close.

Claim. Assuming that the protocol Π_{Com} securely realizes $\mathcal{F}_{\mathsf{Com}}$ producing a statistically hiding view in the presence of a malicious computationally unbounded adversary with adversary structure $\mathcal{Z}_{\mathsf{S}}$, the hybrid distributions H_2 and H_3 are statistically close.

Proof. Let s be the number of executions of Π_{Com} in the protocol where an honest party is the committer. The distributions of H_2 and H_3 can be shown as statistically close by considering the following set of $s + 1$ intermediate hybrid distributions:

– Hybrid $\mathsf{H}'_0 = \mathsf{H}_3$. This is the hybrid distribution where all the inputs used for the honest parties are as in the protocol simulation $\mathsf{Sim}_{\mathsf{S}}$, independent of the real protocol inputs. The view produced here is composed of the view of these commitment protocols, and that of the rest of the real offline protocol created depending on these inputs, with the exception, of course, that all the un-watched executions of the virtual protocol are replaced by simulated views, and the protocol aborts whenever the $\mathsf{Sim}_{\mathsf{S}}$ aborts.

- Hybrid H'_k. For each $k \in [s]$, in this hybrid, in up to the k^{th} execution of the commitment protocol, all the inputs used for the honest parties are as in the real execution and the rest are as in the protocol simulation $\mathsf{Sim}_{\mathsf{S}}$.
- Hybrid $\mathsf{H}'_s = \mathsf{H}_2$. This hybrid distribution contains all honest party inputs and commitment executions as in the real execution of the protocol. The rest of the view of the offline protocol is generated with this as the basis.

Note that for $k \in [s]$, each pair of adjacent hybrids H'_k and H'_{k-1} differ only in that the k^{th} execution of Π_{Com} uses a different input. The rest of the view is generated on its basis, in the same way. Let x be the input to this protocol in the real execution and x' be this input in the simulation. Let ϵ be the statistical difference between the distribution $\{\mathbf{View}_{\Pi_{\mathsf{Com}}}(x)\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$ as in $\mathsf{Sim}_{\mathsf{S}}$ and the real distribution $\{\mathbf{View}_{\Pi_{\mathsf{Com}}}(x')\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$. Then the statistical difference between the adjacent hybrids can be no more than ϵ, which is negligible. It follows from the triangle inequality of statistical differences that the difference between the distributions H_2 and H_3 is $\leq s\epsilon$. Therefore, these distributions are statistically close.

Claim. Assuming that the protocol Π_{Com} securely realizes $\mathcal{F}_{\mathsf{Com}}$ in the presence of a malicious computationally unbounded adversary with adversary structure $\mathcal{Z}_{\mathsf{S}}$, the hybrid distributions H_3 and H_4 are statistically close.

Proof. Let s be the number of executions of Π_{Com} in the protocol in the randomness generation phase. The distributions of H_4 and H_3 can be shown as statistically close by considering the following set of $s+1$ intermediate hybrid distributions:

- Hybrid $\mathsf{H}'_0 = \mathsf{H}_4$. This is the distribution in which all the commitment protocol executions in the randomness generation phase are replaced by executions of $\mathsf{Sim}_{\mathsf{S}}^{\Pi_{\mathsf{Com}}}$.
- Hybrid H'_k. For all $k \in [s]$, in this hybrid, in up to the k^{th} instance of the commitment protocol, the real protocol Π_{Com} is executed, and all other executions are replaced by executions of $\mathsf{Sim}_{\mathsf{S}}^{\Pi_{\mathsf{Com}}}$.
- Hybrid $\mathsf{H}'_s = \mathsf{H}_3$. In this distribution, all the commitment protocol executions in the randomness generation phase are executions of the real protocol Π_{Com}.

Note that for $k \in [s]$, each pair of adjacent hybrids H'_k and H'_{k-1} differ only in the view of the k^{th} execution of Π_{Com} in the randomness generation phase. Let ϵ be the statistical difference between the distribution $\{\mathbf{View}_{\Pi_{\mathsf{Com}}}\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$ as in $\mathsf{Sim}_{\mathsf{S}}$ and the real distribution $\{\mathbf{View}^*_{\Pi_{\mathsf{Com}}}\}_{\kappa \in \mathbb{N}, \vec{r} \in \mathcal{R}^n}$. Then the statistical difference between the adjacent hybrids can be no more than ϵ, which is negligible. It follows from the triangle inequality of statistical differences that the difference between the distributions H_3 and H_4 is $\leq s\epsilon$. Therefore, these distributions are statistically close.

Since we have now shown that hybrid $\mathsf{H}_0 \overset{s}{\equiv} \mathsf{H}_4$ and $\mathsf{H}_4 \equiv \mathsf{H}_5$, it holds that the simulated and the real view of the protocol are statistically close.