Secure Computation with Minimal Interaction, Revisited

Yuval Ishai*, Ranjit Kumaresan**, Eyal Kushilevitz***, and Anat Paskin-Cherniavsky†

Abstract. Motivated by the goal of improving the concrete efficiency of secure multiparty computation (MPC), we revisit the question of MPC with only two rounds of interaction. We consider a minimal setting in which parties can communicate over secure point-to-point channels and where no broadcast channel or other form of setup is available.

Katz and Ostrovsky (Crypto 2004) obtained negative results for such protocols with $n = 2$ parties. Ishai et al. (Crypto 2010) showed that if only one party may be corrupted, then $n \geq 5$ parties can securely compute any function in this setting, with guaranteed output delivery, assuming one-way functions exist. In this work, we complement the above results by presenting positive and negative results for the cases where $n = 3$ or $n = 4$ and where there is a *single malicious party*.

When $n = 3$, we show a 2-round protocol which is secure with "selective abort" against a single malicious party. The protocol makes a black-box use of a pseudorandom generator or alternatively can offer unconditional security for functionalities in NC^1. The concrete efficiency of this protocol is comparable to the efficiency of secure two-party computation protocols for *semi-honest* parties based on garbled circuits.

When $n = 4$ in the setting described above, we show the following:

- A *statistical VSS* protocol that has a 1-round sharing phase and 1-round reconstruction phase. This improves over the state-of-the-art result of Patra et al. (Crypto 2009) whose VSS protocol required 2 rounds in the reconstruction phase.
- A 2-round statistically secure protocol for *linear functionalities* with guaranteed output delivery. This implies a 2-round 4-party fair coin tossing protocol. We complement this by a negative result, showing that there is a (nonlinear) function for which there is no 2-round statistically secure protocol.
- A 2-round computationally secure protocol for *general functionalities* with guaranteed output delivery, under the assumption that injective (one-to-one) one-way functions exist.

---

* Department of Computer Science, Technion. Email: `yuvali@cs.technion.ac.il`. Research supported by the European Union's Tenth Framework Programme (FP10/2010-2016) under grant agreement no. 259426 ERC-CaC, ISF grant 1709/14 and BSF grant 2012378.

** MIT CSAIL. Email: `ranjit@csail.mit.edu`. Supported by Qatar Computing Research Institute. Work done in part while at the Technion.

*** Department of Computer Science, Technion. Email: `eyalk@cs.technion.ac.il`. Research supported by ISF grant 1709/14 and BSF grant 2012378.

† Department of Computer Science, Ariel University. Email: `anps83@gmail.com`

– A 2-round protocol for general functionalities with guaranteed output delivery in the *preprocessing model*, whose correlated randomness complexity is proportional to the length of the inputs. This protocol makes a black-box use of a pseudorandom generator or alternatively can offer unconditional security for functionalities in NC^1.

Prior to our work, the feasibility results implied by our positive results were not known to hold even in the stronger MPC model considered by Gennaro et al. (Crypto 2002), where a broadcast channel is available.

Keywords: Secure multiparty computation, round complexity, efficiency.

1 Introduction

Suppose that two or more parties wish to compute some function on their sensitive inputs while hiding the inputs from each other to the extent possible. One solution would be to employ an external trusted server. Such a trust assumption gives rise to the following minimalist protocol: each party sends its input to the server, who computes the result and sends only the output back to the parties.

However, trusting an external server has several drawbacks, such as being susceptible to server breaches. To eliminate the single point of failure, the parties may employ a secure multiparty computation (MPC) protocol for distributing the trust between the parties. When replacing the external trusted server with an MPC protocol, a major practical disadvantage is that we lose the minimalist structure of the earlier protocol. Indeed, MPC protocols that offer security against malicious parties typically require a substantial amount of interaction. For instance,

– Implementing *broadcast* (a special case of MPC) over secure point-to-point channels generally requires more than two rounds [13].
– Even if broadcast is given for free, 3 or more rounds are necessary for general MPC protocols that tolerate $t \geq 2$ malicious parties and guarantee fairness [16].

Fortunately, neither of the above limitations rules out the possibility of obtaining 2-round MPC protocols secure against a *single malicious party*. This was exploited in the work of Ishai et al. [20], who showed that if only one party can be corrupted, then $n \geq 5$ parties can securely compute any function of their inputs, with guaranteed output delivery, by using only two rounds of interaction over secure point-to-point channels, and *without assuming broadcast or any additional setup*. Since a similar result can be ruled out in the case of $n = 2$ parties [24], the work of [20] leaves open the corresponding question for $n = 3$ and $n = 4$.

This question may be highly relevant to real world situations where the number of parties is small and the existence of two or more corrupted parties is unlikely. Indeed, the only real world deployment of MPC that we are aware of is for the case of $n = 3$ and $t = 1$ (cf. [6, 7]). Furthermore, in settings where secure computation between multiple servers involves long-term secrets, such as cryptographic keys or sensitive databases, it may be preferable to employ three or more servers as opposed to two for the purpose of recovery from faults. Indeed, in secure 2-server solutions the long-term secrets are lost forever if one of the servers malfunctions. Finally, the existence of a strict honest majority allows for achieving stronger security goals, such as fairness and strong forms of composability, that are provably unrealizable in the two-party setting, and moreover it gives hope for designing leaner protocols that use weaker cryptographic assumptions and have better concrete efficiency. Thus, we believe that positive results in this regime (i.e., 2-round protocols for $n = 3$ and $n = 4$) may have strong relevance to the goal of practically efficient secure computation.

Our interest in this problem is motivated not only by the quantitative goal of minimizing the amount of interaction, but also by qualitative advantages of 2-round protocols over protocols with more rounds. For instance, as pointed out in [20], the minimal interaction pattern of 2-round protocols makes it possible to divide the secure computation process into two non-interactive stages of input contribution and output delivery. These stages can be performed independently of each other in an asynchronous manner, allowing clients to go online only when their inputs change, and continue to (passively) receive periodic outputs while inputs of other parties may change.

Our results. We obtain several results on the existence of 2-round MPC protocols over secure point-to-point channels, without broadcast or any additional setup, which tolerate a single malicious party out of $n = 3$ or $n = 4$ parties.

Three-party setting. In an information-theoretic setting without a broadcast channel, the broadcast functionality itself is unrealizable for $n = 3$ and $t = 1$ [25]. Therefore, even if we wish to obtain secure computation protocols with perfect/statistical security, but with *guaranteed output delivery*, then we have to assume a broadcast channel. In the computational setting, broadcast is realizable in two rounds using digital signatures (assuming a public key infrastructure setup). Further, assuming indistinguishability obfuscation and a CRS setup, there exist 2-round protocols which tolerate an arbitrary number of corruptions $t < n$ [14, 2]. These protocols guarantee fairness when $t = 1$ and $n = 3$ (more generally, when $t < n/2$), and also have nearly optimal communication complexity. However, the above computationally secure protocols require a trusted setup and, perhaps more importantly, they rely on strong cryptographic assumptions and have poor concrete efficiency.

Fortunately, as we show, it turns out that a further relaxation of this notion, referred to as "security-with-selective-abort," allows us to obtain statistical security even without resorting to the use of a broadcast channel or a trusted setup. This notion of security, introduced in [18], differs from the standard notion of security-with-abort in that it allows the adversary (after learning its own outputs) to individually decide for each uncorrupted party whether this party will obtain its correct output or will abort with the special output "$\perp$". Our main result in this setting is the following:

- There exists a 2-round, 3-party general MPC protocol over secure point-to-point channels, that provides security-with-selective-abort in the presence of a single malicious party. The protocol provides statistical security for functionalities in NC^1 and computational security for general functionalities by making a black-box use of a PRG.[1]

The above protocol is very efficient in concrete terms. There is a large body of recent work on optimizing the efficiency of 2-party protocols based on garbled circuits. A recent work of Choi et al. [9] considered the 3-party setting, but required security against 2 malicious parties and thus did not offer better efficiency than that of 2-party protocols. Our work suggests that settling for security against a single party can lead to better overall efficiency while also minimizing round complexity. In particular, our 3-party protocol is roughly as efficient as 2-party *semi-honest* garbled circuit protocols. See discussion in Section 3.

Four-party setting. Gennaro et al. [15] show the impossibility of 2-round *perfectly secure* protocols for secure computation for $n = 4$ and $t = 1$, even assuming a broadcast channel. Ishai et al. [20] show a secure-with-selective-abort protocol in this setting over point-to-point channels. Their protocol does not guarantee output delivery. We complete the picture in several different ways. We start by focusing on the simpler question of designing verifiable secret sharing (VSS) protocols. Prior to our work, for the case when $n = 4$ and $t = 1$, it was known that (1) there exists a 1-round sharing and 2-round reconstruction statistical VSS protocol [29], and (2) there exists a 2-round sharing and 1-round reconstruction statistical VSS protocol [1]. We improve the state-of-the-art by showing that:

- There exists a 4-party statistically secure VSS protocol over point-to-point channels that tolerates a single malicious party and requires one round in the sharing phase and one round in the reconstruction phase.

The above result is somewhat unexpected in light of the results from [29, 1], and the corresponding protocol is significantly more involved than other 1-round VSS protocols. Our 1-round VSS protocol implies statistically secure 2-round protocols for fair coin-tossing and simultaneous broadcast over point-to-point channels. More generally, we show that:

- There exists a 2-round 4-party statistically secure MPC protocol for *linear functionalities* (that compute a linear mapping from inputs to outputs) over secure point-to-point channels, providing full security against a single malicious party.

We complement the above positive result by proving the following negative result:

- There exists a nonlinear function which cannot be realized by a protocol as above.

Taken together, the two results above showcase a unique provable separation between the round complexity of linear functionalities (which capture coin-tossing and secure multicast as special cases) and that of higher degree functions. Next, we show that settling for computational security allows us to beat the previous negative result.

[1] Our information-theoretic protocols are limited to NC^1 like all known constant-round protocols, even in the semi-honest model. However, settling for computational security, all our protocols apply to general circuits by using any PRG as a black box.

- Assuming the existence of injective (one-to-one) one-way functions, there exists a 2-round 4-party *computationally* secure MPC protocol for *general functionalities* over secure point-to-point channels, providing full security against a single malicious party.

None of our previous results require a setup assumption. A natural question is whether it is possible to obtain statistical security (at least for functionalities in NC^1) in the same setting by relying on some form of setup. Several prior works [5, 10, 11, 19, 8] obtain information-theoretic security in a so-called preprocessing model, where the parties are given access to a source of correlated randomness before the inputs are known. However, these protocols either have a higher round complexity, or alternatively make use of correlated randomness whose size grows exponentially with the input length [19, 4]. We present a protocol in this setting where the size of correlated randomness is exactly the length of the inputs. Formally, we show in Appendix G that:

- Assuming a correlated randomness setup, there exists a 2-round 4-party MPC protocol over secure point-to-point channels, providing full security against a single malicious party. The protocol provides statistical security for functionalities in NC^1 and computational security for general functionalities by making a black-box use of a PRG. The size of the correlated randomness is linear in the input size.

Prior to our work, our positive results in either the 3-party or 4-party settings were not known to hold even in the setting considered where a broadcast channel is available, which was studied in the line of work originating from [15, 16]. Moreover, our protocols are secure against adaptive and rushing adversaries. Finally, while we analyze our protocols in the standalone setting, they are in fact composable (in particular, none of our simulators is rewinding). Table 1 (Appendix A) summarizes our results.

Technical overview. We now give a very brief and high level overview of some of our results. The main primitives that we use in our protocols are private simultaneous message (PSM) protocols [12] and 1-private secret sharing schemes (cf. Section 2). Our high level strategy is similar to the one used in [20]. The parties secret share their inputs among other parties in the first round. Then in the second round they make use of PSM subprotocols to reconstruct parties' inputs from the shares, and also to evaluate a function on the reconstructed inputs. Given the above, there are still two main issues that need to be resolved: (1) a malicious PSM client may supply inconsistent shares of honest parties inputs inside the PSM, and (2) a malicious party may supply inconsistent shares of its own input to honest parties. Thus different PSM instances may reconstruct different inputs thereby generating different outputs all of which seem correct.

Ishai et al. [20] get around (1) & (2) by using $(n-2)$-client PSM. Note that for $n \geq 5$ there are at least two honest clients and these two clients hold *all* the shares of all parties. Thus, it is easy to detect inconsistent input shares *inside* the PSM, and it is possible to either apply a "correction" inside the PSM or easily ensure that incorrect PSM outputs are discarded. In our setting, i.e., $n \in \{3, 4\}$, we have to deal with 2-client PSMs. This is obviously necessary when

$n = 3$. We can use 3-client PSM when $n = 4$, but this PSM cannot be expected to deliver output since a malicious client can simply abort this PSM. For these reasons, techniques from [20] do not work when $n \in \{3, 4\}$. We can no longer apply corrections inside the PSM or easily identify incorrect PSM outputs.

To get around (1), we use a novel "view reconstruction" technique (cf. Section 3). When $n = 3$, this technique suffices, together with some additional ideas, to get around both (1) & (2). To get around (2) when $n = 4$, we use information-theoretic MACs for secure linear function evaluation and non-interactive commitments for general secure function evaluation. Additional complications arise when using MACs inside the PSM and we overcome these by employing a cut-and-choose technique (cf. Section 4).

2 Preliminaries

In this section, we provide definitions of verifiable secret sharing and private simultaneous message protocols. We also give descriptions of secret sharing schemes we use. We refer to Appendix B for more details.

Verifiable secret sharing (VSS). In this work, we focus on the statistical variant of verifiable secret sharing. We give the general definition below, but will construct protocols for the specific case of $n = 4$ and $t = 1$.

Definition 1. *Let σ be a statistical security parameter. A two-phase protocol for parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, where a distinguished dealer $D \in \mathcal{P}$ holds initial input $s \in \mathbb{F}$, is a* statistical VSS protocol tolerating t malicious parties *if the following conditions hold for any adversary controlling at most t parties:*

Privacy *If the dealer is honest at the end of the first phase (the* sharing phase*), then at the end of this phase the joint view of the malicious parties is independent of the dealer's input s.*

Correctness *Each honest party P_i outputs a value s_i at the end of the second phase (the* reconstruction phase*). If the dealer is honest, then except with probability negligible in σ, it holds that $s_i = s$.*

Commitment *Except with probability negligible in σ, the joint view of the honest parties at the end of the sharing phase defines a value s' such that $s_i = s'$ for every honest P_i.* ♢

The PSM model. A private simultaneous messages (PSM) protocol [12] is a non-interactive protocol involving m parties $P_1, \ldots, P_m$, who share a common random string $r = r^{\mathrm{psm}}$, and an external referee who has no access to r. In such a protocol, each party P_i sends a single message to the referee based on its input x_i and r. These m messages should allow the referee to compute some function of the inputs without revealing any additional information about the inputs. Our definitions below are taken almost verbatim from [20].

Formally, a PSM protocol π for a function $f : \{0,1\}^{\ell \times m} \rightarrow \{0,1\}^*$ is defined by $R(\ell)$, a randomness length parameter, m message algorithms $A_1, \ldots, A_m$ and a reconstruction algorithm Rec, such that the following requirements hold.

- *Correctness:* for every input length ℓ, all $x_1, \ldots, x_m \in \{0,1\}^\ell$, and all $r \in \{0,1\}^{R(\ell)}$, we have $\mathsf{Rec}(A_1(x_1, r), \ldots, A_m(x_m, r)) = f(x_1, \ldots, x_m)$.
- *Privacy:* there is a simulator $\mathcal{S}_\pi^{\text{trans}}$ such that, for all $x_1, \ldots, x_m$ of length ℓ, the distribution $\mathcal{S}_\pi^{\text{trans}}(1^\ell, f(x_1, \ldots, x_m))$ is indistinguishable from $(A_1(x_1, r), \ldots, A_m(x_m, r))$.

We consider either perfect or computational privacy, depending on the notion of indistinguishability. (For simplicity, we use the input length ℓ also as security parameter, as in [17]; this is without loss of generality, by padding inputs to the required length.)

A *robust* PSM protocol π should additionally guarantee that even if a subset of the m parties is malicious, the protocol still satisfies a notion of "security with abort." That is, the effect of the messages sent by corrupted parties on the output can be simulated by either inputting to f a valid set of inputs (independently of the honest parties' inputs) or by making the referee abort. This is formalized as follows.

- *Statistical robustness:* For any subset $T \subset [m]$, there is an efficient (black-box) simulator $\mathcal{S}_\pi^{\text{ext}}$ which, given access to the common r and to the messages sent by (possibly malicious) parties P_i^*, $i \in T$, can generate a distribution x_T^* over x_i, $i \in T$, such that the output of Rec on inputs $A_T(x_T^*, r), A_{\overline{T}}(x_{\overline{T}}, r)$ is statistically close to the "real-world" output of Rec when receiving messages from the m parties on a randomly chosen r. The latter real-world output is defined by picking r at random, letting party P_i pick a message according to A_i, if $i \notin T$, and according to P_i^* for $i \in T$, and applying Rec to the m messages. We allow $\mathcal{S}_\pi^{\text{ext}}$ to produce a special symbol $\perp$ (indicating abort) on behalf of some party P_i^*, in which case Rec outputs $\perp$ as well.

The following theorem summarizes some known facts about PSM protocols.

Theorem 1 ([12, 20, 28]). *(i) For any $f \in \mathrm{NC}^1$, there is a polynomial-time, perfectly private, and statistically robust PSM protocol. (ii) For any polynomial-time computable f, there is a polynomial-time, computationally private, and statistically robust PSM protocol which uses any PRG as a black box.*

Secret sharing. In a t-private n-party secret sharing scheme every t parties learn nothing about the secret, and every $t+1$ parties can jointly reconstruct it. A secret sharing scheme is *efficiently extendable*, if for any subset $T \subseteq [n]$, it is possible to efficiently check whether the (purported) shares to T are consistent with a valid sharing of some secret s. Additionally, in case the shares are consistent, it is possible to efficiently sample a (full) sharing of some secret which is consistent with that partial sharing. In our protocols, we use 2-out-of-2 additive secret sharing and 1-private 3-party CNF secret sharing.

Additive sharing. In 2-out-of-2 additive sharing over $\mathbb{F}_2$, given both shares r_1, r_2, we can reconstruct the secret as $s = r_1 \oplus r_2$. On the other hand, given the secret s and one of the shares r_1, we can determine the remaining share $r_2 = s \oplus r_1$.

CNF sharing [21]. In 1-private 3-party CNF sharing over $\mathbb{F}_2$, we choose random $r_1, r_2 \in \mathbb{F}_2$, compute $r_3 = s \oplus r_1 \oplus r_2$, and set the CNF shares held by

P_1, P_2, P_3 as $\langle r_2, r_3\rangle, \langle r_3, r_1\rangle, \langle r_1, r_2\rangle$ respectively. Given two of the three CNF shares, say $\langle r_1, r_2\rangle, \langle r_2, r_3\rangle$ we can reconstruct the secret $s = r_1 \oplus r_2 \oplus r_3$. Also, given s and one of the shares say $\langle r_1, r_2\rangle$, we can determine the remaining shares as $\langle r_2, s\oplus r_1 \oplus r_2\rangle$ and $\langle s\oplus r_1\oplus r_2, r_1\rangle$. We say that P_1, P_2 hold "consistent" CNF shares if P_1, P_2 respectively hold $\langle r_2, r_3\rangle, \langle r_3', r_1\rangle$ with $r_3' = r_3$.

Notation. We let n denote the number of parties. Note in this paper $n \in \{3, 4\}$. The notation T_i (resp. $T_{i,j}$) denotes the set $[n] \setminus \{i\}$ (resp. $[n] \setminus \{i, j\}$), where the value of n is clear from the context. Throughout this paper, the number of corrupt parties $t = 1$. Since this is the case, we sometimes abuse notation and use t as a variable to denote parties' index (e.g., P_t). We let $r_{i,j}^{\mathrm{psm}} = r_{j,i}^{\mathrm{psm}}$ to denote the shared randomness for PSM executions involving clients P_i and P_j.

3 2-Round 3-Party Computation with Selective Abort Security

Recall that in security with selective abort, the adversary is able to deny output to an honest party (i.e., there is no guaranteed output delivery), and further it can choose to do so individually for each honest party. We wish to stress that the abort is dependent only on the inputs/outputs of the corrupt party and is otherwise (statistically) independent of the inputs/outputs of the honest parties.

A first attempt. Consider the following protocol which makes use of additive sharing and PSM subprotocols. Each party P_i first additively shares its input x_i into $x_{i,j}$ and $x_{i,k}$ (i.e., $x_i = x_{i,j} \oplus x_{i,k}$) and sends $x_{i,j}$ to party P_j and $x_{i,k}$ to party P_k. In the second round, parties execute pairwise (robust) PSMs that first reconstruct each party's input from the additive shares possessed by the PSM clients, and then compute the output from the reconstructed inputs. It should be clear that the above yields a secure protocol in the semi-honest setting.

Predictably, things go wrong in the presence of a malicious adversary. Specifically, an adversary that corrupts, say, P_1 can carry out the following attack: Party P_1 can use input 0 in the PSM execution where P_1 and P_2 are the PSM clients and P_3 is the PSM referee. Then, P_1 uses a different input, say 1 in the PSM execution where P_1 and P_3 are the PSM clients and P_2 is the PSM referee. This results in the undesirable situation where P_2 and P_3 disagree on the output and, furthermore, are not even aware that there may be a disagreement. Note that this does not yield security with selective abort, since honest parties accept outputs that are computed using different values for the corrupt input. In other words, there is no single effective corrupt input (to be extracted by the 'simulator' in the ideal execution) that explains all honest outputs. To counter this attack, we employ the following "view reconstruction trick."

View reconstruction trick. Essentially this trick tries to reconstruct the (first round) view of the PSM referee using the views supplied by the PSM clients. Note that the "view" in the naïve protocol described above consists of additive shares supplied by the parties. Fortunately, the *efficient extendability* of linear secret sharing schemes such as the additive secret sharing and CNF secret sharing, enables us to reconstruct the unique share that must be held by the PSM referee. (For more details see Section 2 and [20].)

To see this trick in action, consider a concrete example. Suppose P_i and P_j are PSM clients and P_k is the PSM referee. Note that P_k's view consists of the shares $x_{i,k}$ sent by P_i and $x_{j,k}$ sent by P_j. Now in the PSM subprotocol (instantiated in the naïve protocol) suppose party P_i supplies input x'_i and party P_j supplies input x'_j. (If P_i (resp. P_j) is not honest then $x'_i = x_i$ (resp. $x'_j = x_j$) may not hold.) In the PSM protocol, we now ask P_i to supply in addition to its input $x'_i = x_i$ also the shares obtained in round 1, namely $x'_{j,i} = x_{j,i}$ obtained from P_j and $x'_{k,i} = x_{k,i}$ obtained from P_k. We ask P_j to do the same as well, i.e., P_j supplies $x'_j = x_j$, $x'_{i,j} = x_{i,j}$, $x'_{k,j} = x_{k,j}$. Of course, a malicious party, say P_i, may not supply the correct inputs or shares as it obtained from the honest parties (i.e., it may be the case that $x'_i \neq x_i$ or $x'_{j,i} \neq x_{j,i}$ or $x'_{k,i} \neq x_{k,i}$). Anyway, we can compute the values that *ought* to be held by P_k using the values supplied by P_i and P_j. For instance, the values $x_{k,i}, x_{k,j}$ can directly be obtained from P_i, P_j since they supplied $x'_{k,i}, x'_{k,j}$ (respectively) to the PSM subprotocol. The values $x_{i,k}$ (resp. $x_{j,k}$) can be reconstructed as $x'_i \oplus x'_{i,j}$ where x'_i was supplied by P_i and $x'_{i,j}$ was supplied by P_j.

In our modified protocol, we let the PSM referee, say P_k to accept the final output only if the reconstructed view from the PSM protocol matches its first round view, i.e., only if $x'_{k,i} = x_{k,i}$, $x'_{k,j} = x_{k,j}$, $x'_{i,k} = x_{i,k}$, and $x'_{j,k} = x_{j,k}$ all hold. We prove the following theorem.

Theorem 2. *There exists a 2-round 3-party secure-with-selective-abort protocol for secure function evaluation over point-to-point channels that tolerates a single malicious party. The protocol provides statistical security for functionalities in* NC^1 *and computational security for general functionalities by making a black-box use of a pseudorandom generator.*

Proof. The formal protocol is described in Figure 1. We provide a sketch of the simulation and the analysis below. See Appendix C for the full proof.

<u>*Simulation sketch.*</u> Denote the corrupt party by P_ℓ. Let P_i, P_j be the remaining (honest) parties. The simulator begins by sending random additive shares to the corrupt party on behalf of the honest parties. It also sends and receives randomness to be used in the PSM executions in the next round. Note that the simulator also receives additive shares from the corrupt party. Using the additive shares, the simulator computes the effective input say $\hat{x}_\ell$ of the corrupt party (i.e., by simply xor-ing the additive shares). Then, the simulator sends $\hat{x}_\ell$ to the trusted party first, and obtains the output z_ℓ.

Next the simulator invokes the PSM simulator $\mathcal{S}^{\text{trans}}_{\pi_{i,j}}$ (guaranteed by the privacy property) on inputs z_ℓ and the additive shares sent on behalf of the honest parties. Denote the output of the $\mathcal{S}^{\text{trans}}_{\pi_{i,j}}$ by $\tau_{i,\ell}$ and $\tau_{j,\ell}$. Acting as the honest party P_i (resp. P_j), the simulator sends $\tau_{i,\ell}$ (resp. $\tau_{j,\ell}$) to the corrupt party. It remains to be shown how the simulator decides which uncorrupted parties learn the output and which receive $\perp$. To do this, the simulator does the following. First, acting as the honest party P_i the simulator receives the

PSM message $\tau_{\ell,i}$ that P_ℓ sends to P_i as part of PSM execution $\pi_{\ell,j}$. Similarly, acting as P_j, the simulator also receives $\tau_{\ell,j}$. Next, the simulator invokes the PSM simulator $\mathcal{S}^{\text{ext}}_{\pi_{\ell,i}}$ on the PSM message $\tau_{\ell,i}$ (and also the PSM randomness) to decide what effective input P_ℓ used in PSM subprotocol $\pi_{\ell,j}$. Depending on this input, the simulator then decides whether P_i will accept the output of $\pi_{\ell,j}$ or not. Specifically as in the real execution, the simulator checks if the shares input by P_ℓ are consistent with those held by P_i. If this is indeed the case, then the simulator asks the trusted party to deliver output to P_i, else it asks the trusted party to deliver $\perp$ to P_i. Whether P_j gets the output or not is also handled similarly by the simulator.

Analysis sketch. We first consider a hybrid experiment which is exactly the same as the real execution except that the PSM messages sent by the honest parties to P_ℓ are replaced by the simulated PSM transcripts generated by $\mathcal{S}^{\text{trans}}_{\pi_{i,j}}$. To generate these transcripts we first extract the input $\hat{x}_\ell$ by xor-ing the additive shares sent by P_ℓ, and then compute the output of $\pi_{i,j}$ using inputs provided by honest parties and $\hat{x}_\ell$. We then supply this output to $\mathcal{S}^{\text{trans}}_{\pi_{i,j}}$ to generate the simulated PSM transcripts. The privacy property of the PSM protocol implies that the joint distribution of the view of the adversary and honest outputs in the real protocol is indistinguishable from the corresponding distribution in the hybrid execution.

Note that the distribution of the additive shares and the PSM randomness sent by the simulator in the ideal execution is identical to the distribution of the corresponding values in the hybrid execution. Thus, to prove indistinguishability of the hybrid execution and the ideal execution it suffices to focus on the distribution of honest outputs. Note that in the ideal execution the honest outputs are generated using the true honest inputs and extracted input $\hat{x}_\ell$.

We first show that honest party P_i (resp. P_j) that accepts a non-$\perp$ output in the hybrid execution is ensured that this output is computed using the true honest inputs and the corrupt input $\hat{x}_\ell$. It is here that we use the view reconstruction trick. Specifically now, (1) if P_ℓ supplied incorrect input, then the reconstructed share $x'_{\ell,i}$ (which is revealed as part of the output of $\pi_{\ell,j}$) does not equal $x_{\ell,i}$ possessed by P_i and thus the final output is rejected, and (2) if P_ℓ supplied inconsistent share $x'_{i,\ell} \neq x_{i,\ell}$ inside $\pi_{\ell,j}$, then since this value is revealed as part of the output of $\pi_{\ell,j}$, the final output will be rejected by P_i.

Given the above it remains to be shown that the set of honest parties that receive $\perp$ in the ideal execution equals the set of honest parties that output $\perp$ in the hybrid execution. To prove the above, we use the fact that for all $j \in T_\ell$, with all but negligible probability the PSM simulator $\mathcal{S}^{\text{ext}}_{\pi_{\ell,j}}$ extracts the input supplied by P_ℓ in the PSM execution $\pi_{\ell,j}$. It follows by simple inspection that the criterion used to add i to S_ℓ in the simulation is essentially the same as the criterion used by P_i to reject the final output of $\pi_{\ell,j}$ in the hybrid execution. □

Concrete efficiency. Robust PSM subprotocols can be based on Yao garbled circuits [12, 28]. The concrete cost of such a robust PSM protocol is essentially the same as a single Yao garbled circuit and incurs an additional cost proportional to the length of the inputs (and is otherwise independent of the complexity

Round 1.

- For $i \in [3]$, each P_i additively shares its input x_i into $x_{i,j}$ and $x_{i,k}$, and sends $x_{i,j}$ to P_j, and $x_{i,k}$ to P_k for distinct $j, k \in [3] \setminus \{i\}$.
- Every pair of parties P_i, P_j, $i, j \in [3]$ and $i < j$, exchange randomness $r^{\mathrm{psm}}_{i,j}$. (For instance, by letting P_i pick $r^{\mathrm{psm}}_{i,j}$ and send $r^{\mathrm{psm}}_{i,j}$ to P_j.)

Round 2.

- Every pair of parties P_i and P_j, $i, j \in [3]$ and $i < j$, use shared randomness $r_{i,j}$ to execute a robust PSM protocol $\pi_{i,j}$, that
  - takes input $\tilde{x}_i = (x'_{k,i}, x'_i, x'_{j,i})$ from P_i where $x'_{k,i} = x_{k,i}, x'_i = x_i, x'_{j,i} = x_{j,i}$,
  - takes input $\tilde{x}_j = (x'_{k,j}, x'_j, x'_{i,j})$ from P_j where $x'_{k,j} = x_{k,j}, x'_j = x_j, x'_{i,j} = x_{i,j}$,
  - reconstructs $x'_k = x_{k,i} \oplus x_{k,j}$,
  - computes $z'_k = f_k(x'_1, x'_2, x'_3)$, $x'_{i,k} = x'_i \oplus x_{i,j}$, $x'_{j,k} = x'_j \oplus x_{j,i}$, and
  - delivers output $(z'_k, x'_{i,k}, x'_{j,k}, x'_{k,i}, x'_{k,j})$ to P_k for $k \in [3]$ and $k \notin \{i, j\}$.

Output. Each P_k outputs z'_k if $x'_{i,k} = x_{i,k}$, $x'_{j,k} = x_{j,k}$, $x'_{k,i} = x_{k,i}$, and $x'_{k,j} = x_{k,j}$ hold, else it outputs $\perp$.

Fig. 1. 2-round 3-party secure-with-selective-abort protocol.

of f). Thus our 3-party protocol costs essentially the same as cost of transmitting and evaluating 3 garbled circuits, i.e., thrice the cost of semi-honest 2-party Yao. Contrast this with the concrete cost of realizing state-of-the-art malicously secure *two-party* protocols which is essentially the cost of transmitting and evaluating roughly σ garbled circuits where σ denotes the statistical security parameter. We previously argued that 3-party protocols provide more redundancy and stability compared to 2-party protocols. Now by settling for just security-with-selective-abort, our three-party protocol provides a much better alternative from a cost perspective as well. All this is in addition to the fact that our 3-party protocol requires only two rounds over point-to-point channels. In contrast, current implementations of 3-party protocols [6, 7] require rounds proportional to the depth of the circuit, provide only semi-honest security, or require use of broadcast.

4 4-Party Statistical VSS in a Total of 2 Rounds

Let the set of parties be $\{D, P_1, P_2, P_3\}$. First, let us look at a naïve protocol that assumes the existence of a broadcast channel. Here, the dealer CNF shares its input in the sharing phase. Then in the reconstruction phase, parties simply broadcast the CNF shares they obtained from the dealer. To decide on the output, parties construct an "inconsistency graph" G which tells which parties broadcasted consistent CNF shares.

Sharing Phase. The dealer CNF shares (according to a 1-private 3-party CNF scheme) its secret s among P_1, P_2, P_3. That is, it chooses random s_1, s_2, s_3 subject to $\bigoplus_{i=1,2,3} s_i = s$, and sends CNF share $\{s_j\}_{j \neq i}$ to party P_i for $i \in [3]$.

Reconstruction Phase. Each party P_i broadcasts its share $\{s^{(i)}_j = s_j\}_{j \neq i}$.

Local Computation. D outputs s and terminates the protocol. For every $j,k \in [3]$, define $\mathsf{rec}_{j,k} = s_j^{(k)} \oplus \bigoplus_{i \neq j} s_i^{(j)}$ (i.e., secret reconstructed from CNF shares possessed by P_j and P_k). Let G denote the 3-vertex inconsistency graph which contains an edge between vertices $i,j \in [3]$ iff $\exists k \in [3] \setminus \{i,j\}$ such that $s_k^{(i)} \neq s_k^{(j)}$. (That is, P_i and P_j disagree on the share s_k.)

- (Single-edge case) If G contains exactly one edge, output $\perp$.
- (Even-edge case) Else, if $\exists (j,k) \notin G$, then each party outputs $\mathsf{rec}_{j,k}$.
- (Triple-edge case) If there is no such j,k, then output default value say $\perp$.

It can be easily shown that the above protocol works as long as G does not contain exactly one edge. (See Appendix D.) The difficulty in handling the single-edge case comes because parties do not know which of the inconsistent CNF shares to trust, i.e., which of $s_k^{(i)} \neq s_k^{(j)}$ when $(i,j) \in G$. In the computational setting, this is solved by a trivial use of signatures. In the information-theoretic setting, we can substitute signatures with information-theoretic MACs, but this is not sufficient since such MACs do not have public verification. Fortunately, a combination of MACs with a cut-and-choose technique helps us in this case.

Protocol overview. The high level idea is to use MACs and then apply the cut-and-choose technique to ensure that (1) parties reveal their true share when D is honest, and (2) detect an inconsistent sharing by a dishonest D. In more detail, now we require D to send, in addition to the CNF shares, also authentication information in the form of information-theoretic MACs (such that a forgery is possible only with probability $\mathsf{negl}(\sigma)$). Specifically for each CNF share s_j, the dealer D sends s_j along with σ MAC values $\{M_{j,\ell}^{(i)}\}_{\ell \in [\sigma]}$ to each party P_i for each $j \neq i$, while each party P_j receives the corresponding keys $\{K_{j,\ell}^{(i)}\}_{\ell \in [\sigma]}$ for each $i \neq j$. Each share is authenticated multiple times to allow application of the cut-and-choose technique.

The reconstruction phase is modified to handle, in particular, the case when the inconsistency graph contains exactly one edge. (All other cases are handled exactly as in the naïve attempt described above.) Now we ask each P_i to broadcast its CNF share $\{s_j^{(i)}\}_{j \neq i}$ (as in the naïve construction), and in addition broadcast its MAC values $\{M_{j,\ell}^{(i)}\}_{j \neq i, \ell \in [\sigma]}$. Also we ask each party P_j to pick for every $i \neq j$, a random subset $S_{j,i} \subset [\sigma]$ (this corresponds to the check set for the cut-and-choose step), and send (1) keys $K_{j,\ell}^{(i)}$ for $\ell \in S_{j,i}$ to P_i, and (2) all keys (i.e., $K_{j,\ell}^{(i)}$ for all $\ell \in [\sigma]$) to P_k where $k \in [3] \setminus \{i,j\}$.

Now we explain in more detail how the cut-and-choose technique helps to resolve the single-edge case. Let $(i,j) \in G$ and let $k \notin \{i,j\}$. We consider two cases depending on whether D is honest or not. Note that in either case, we are assured that P_k is honest, and in fact, our protocol will use MAC keys held by P_k to anchor the parties' output towards the correct output. First consider the case when D is honest. Wlog assume P_i is dishonest, and that P_i disagrees with P_j on the value s_k that is supposed to be held by both of them. Note that while P_k does not hold s_k, it does hold the keys $\{K_{k,\ell}^{(i)}\}_{\ell \in [\sigma]}$ to verify the MACs that P_i possesses. Note that the protocol asks P_i to broadcast all its MACs on s_k, and P_k

to send half its keys, say corresponding to some subset $S_{k,i} \subset [\sigma]$, to P_i and all its keys to P_j. While a rushing P_i can wait to receive (half) the keys from P_k to allow forging the corresponding MACs, note that it cannot forge the MACs for the remaining half (except with negligible probability) for which it simply does not know the keys. In other words, when P_i tries to reveal $s'_k \neq s_k$ along with MACs $\{\widetilde{M}^{(i)}_{k,\ell}\}_{\ell \in [\sigma]}$, then with high probability the MAC verification will fail for *all* keys that P_i does not know. Thus, by asking honest P_j and P_k to accept P_i's reveal only if MACs revealed by P_i is consistent with all keys in $\{K^{(i)}_{k,\ell}\}_{\ell \in S_{k,i}}$ (i.e., those that were sent to P_i) and at least one key in $\{K^{(i)}_{k,\ell}\}_{\ell \notin S_{k,i}}$ (i.e., those that were *not* sent to P_i), we are ensured (except with negligible probability) that P_i's reveal $s'_k \neq s_k$ will be rejected by P_j and P_k. Finally note that honest P_j's share s_k is always accepted by the honest parties.

Next, consider the case when D is dishonest. In this case, a single-edge in the inconsistency graph is induced by the inconsistent shares dealt to P_i, P_j. Therefore, the main challenge here is to ensure that all parties agree that D dealt inconsistent shares (as opposed to suspecting that one of the honest parties is deviating from the protocol). Once again, the keys held by P_k serve to anchor all honest parties' decisions on whether to accept or reject reveals made by P_i, P_j. The crux of the argument is the following: except with negligible probability, all parties P_i, P_j, P_k unanimously agree on their decision to accept/reject each of P_i, P_j's reveals. Before we show this, observe that this suffices to achieve resilience against a malicious D. For e.g., suppose both parties' reveals get accepted then if they revealed inconsistent values then all parties agree to output some default value. The case when both parties' reveals get rejected is handled similarly. Finally, when only one of P_i, P_j's reveal is accepted, then all parties can simply agree to output the value corresponding to the reveal that got accepted.

Now we argue that except with negligible probability, all parties will unanimously agree on whether to accept or reject reveals made by P_i, P_j. First observe that the reveals made by a party, say P_j, are either unanimously accepted or unanimously rejected by both P_i and P_k. This is because both P_i and P_k make decisions using the same algorithm on the same values. Next, in our protocol, P_j will accept or reject its own reveal by checking whether its reveal is consistent with the keys that P_k sent to it (i.e., those corresponding to the subset $S_{k,i}$). Thus, if P_j's reveal is rejected by P_j itself, then obviously it will also be rejected by P_i and P_k. Therefore, by way of contradiction, wlog assume that P_j's reveal is rejected by P_i, P_k while it is accepted by P_j. Clearly this happens only if P_k chooses its random subset $S_{k,j}$ such that *all* the MAC values held by P_j corresponding to $S_{k,j}$ are consistent with the keys held by P_k, while *all* the MAC values held by P_j corresponding to $[\sigma] \setminus S_{k,j}$ are *not* consistent with the keys held by P_k. Obviously such an event happens with probability $\binom{\sigma}{\sigma/2}^{-1} = \mathsf{negl}(\sigma)$. Hence we have that with all but negligible probability, all parties P_i, P_j, P_k unanimously agree whether to accept/reject reveals made by P_i and P_j. As explained before, this suffices to prove that agreement holds even when D is dishonest. Fortunately, we can remove the use of broadcast channel in the above protocol. In Appendix D, we prove the following theorem.

Theorem 3. *There exists a 4-party statistically secure protocol for VSS over point-to-point channels that tolerates a single malicious party and requires one round in the sharing phase and one round in the reconstruction phase.*

5 2-Round 4-Party Statistically Secure Computation for Linear Functions Over Point-to-Point Channels

Overview. In the first round of the protocol parties verifiably secret share their inputs (using the protocol from the previous section), and also exchange randomness for running pairwise (robust) PSM executions. Loosely speaking, the PSM executions serve two purposes: (1) parties can evaluate the function on their inputs while preserving privacy, and (2) parties can learn the inconsistency graph corresponding to each VSS sharing. To do (1), the PSM protocol first attempts to reconstruct parties' inputs from the CNF shares held by the PSM clients, and if successful, evaluates the function on these inputs. To do (2), the PSM protocol makes use of the "view reconstruction trick." Note that in the case of VSS, learning the inconsistency graphs was trivial, since parties would broadcast their shares during the reconstruction phase. Unlike VSS, here it is important to protect privacy of these shares throughout the computation. The view reconstruction trick enables us to construct the inconsistency graphs while preserving privacy of the shares.

Recall that each party could potentially receive PSM outputs from three PSM executions. Computing the final output from these PSM outputs is not straightforward, and we will need the inconsistency graphs (generated using outputs of the PSM protocols) to help us. To explain how this is done, we will adopt the perspective of the simulation extraction procedure. Let $m \in [4]$ denote the index of the corrupt party. The extraction procedure constructs the inconsistency graph G' adding edges between vertices if the CNF shares held by corresponding parties are not consistent. If the graph contains all three edges, then the effective input used in this case is 0. We call this the *identifiable triple-edge* case since it is clear that P_m is corrupt. Next, if the graph contains two edges or no edges (i.e., an even number of edges), then we are now assured that there exists a pair of (honest) parties that hold consistent CNF shares of P_m's input. In this case, we can extract the effective input as the secret reconstructed from these consistent CNF shares. We call this case the *resolvable even-edge* case. As was the case in VSS, if G' contains a single-edge then the procedure performs a vote computation step using the MAC values and the corresponding keys. This is to find out which of the two parties is supported by P_m. If there is a unique party that is supported by P_m, then the inconsistency in CNF shares is resolved by using the CNF share possessed by this party. We call this the *resolvable single-edge* case. On the other hand if there is no unique party supported by P_m, then it is clear that P_m is corrupt. We call this the *identifiable single-edge* case. In this case, we extract the effective input used for P_m as the xor of all unique shares (including the inconsistent CNF shares) possessed by all remaining parties.

Observe that the extraction procedure is identical to the VSS extraction procedure except in the identifiable single-edge case. In VSS, it was possible to simply output 0 in the identifiable single-edge case. Here we are not able to replace the corrupt party's input by 0 and then evaluate the function while simultaneously preserving privacy of honest inputs. However, if we use the effective input extracted as described above, then we can exploit the linearity of f to force parties' outputs to be consistent with the extracted input.

Clearly we are done if we force honest parties' outputs in the real protocol to be consistent with the corrupt input extracted by the simulator while preserving privacy of honest parties' inputs. The main obstacle in the implementation is that different honest parties' may hold different inconsistency graphs. The challenge therefore is to design an output computation procedure that allows honest parties' to end up with the same correct output even though they may possess different inconsistency graphs. Also, unlike VSS, here we do not have the luxury of a reconstruction phase where parties can freely disclose their secret shares.

Our output computation procedure makes use of the view reconstruction trick to help each party compute its inconsistency graph, and adapts the cut-and-choose idea from our VSS protocol to help compute the votes (which we can ensure whp that parties agree on). In addition, our procedure exploits the linearity of f to compute the correct output in the identifiable single-edge case. To ensure parties' compute the same output in the resolvable cases, we make use of an "accusation graph" which parties use to determine a pair of honest parties that hold consistent shares of the corrupt input extracted by the simulation procedure described above. Our actual protocol is somewhat nontrivial, and we give a detailed step-by-step overview of the protocol along with the intuition behind the design in Appendix E.2. In Appendix E.3 we prove the following:

Theorem 4. *There exists a 2-round 4-party statistically secure protocol for secure* linear *function evaluation over point-to-point channels that tolerates a single malicious party.*

5.1 Impossibility of 2-Round Statistically Secure 4-Party Computation

In this section, we prove the following:

Theorem 5. *There exists a function which cannot be information-theoretically realized by a 2-round 4-party protocol over point-to-point channels that tolerates a single corrupt party.*

Proof. Assume by way of contradiction that there exists a 2-round statistically secure 4-party protocol π for general secure computation. Let us further set up some notation related to protocol π. Let $A_{i,j}^{(r)}$ denote the algorithm specified by protocol π that is to be executed by (honest) party P_i to generate its r-th round message to P_j. We use the notation

$$m_{i,j}^{(r)} \leftarrow A_{i,j}^{(r)}(x_i, \{\{m_{k,i}^{(s)}\}_{k \in K_i^{(s)}}\}_{s\ :\ 0<s<r}; \omega_i)$$

where x_i (resp. ω_i) represents P_i's input (resp. internal randomness), and $m_{i,j}^{(r)}$ represents P_i's message to P_j in round r, and $K_i^{(s)}$ represents the subset of parties from which P_i receives a message in round s. Wlog, we assume that algorithm $A_{i,i}^{(3)}$ computes the final output of honest P_i.

The function that we consider is a simple non-linear function and is inspired by the oblivious transfer functionality. Let f be such that $f(b, \perp, \perp, (y_0, y_1)) = (y_b, \perp, \perp, \perp)$. That is, f takes as input a bit $b \in \{0, 1\}$ from P_1 and a pair of bits $y_0, y_1 \in \{0, 1\}$ from P_4, and returns y_b to P_1. The parties P_2, P_3 supply no inputs, and parties P_2, P_3, P_4 receive no outputs.

The high level strategy is to launch an attack on the real protocol that cannot be simulated in the ideal execution. We let P_1 be the corrupt party, and show that it can obtain *both* y_0 and y_1 in the real protocol with non-negligible probability. Clearly, no ideal process adversary can do the same, and hence the negative result is establised. At a high level, the adversarial strategy of P_1 is to set things up such that the joint view of P_2 and P_4 would infer that P_1's input is 0, while the joint view of P_3 and P_4 would infer that P_1's input is 1. To do this, P_1 chooses internal randomness ω_1 and computes its first round messages $\tilde{m}_{1,2}^{(1)}, \tilde{m}_{1,4}^{(1)}$ to send to P_2 and P_4 assuming that its input equals 0. Then, it samples uniform randomness $\tilde{\omega}$ such that its first round message to P_4 computed assuming input 1 and randomness $\tilde{\omega}$ matches $\tilde{m}_{1,4}^{(1)}$. Since we are in the information-theoretic regime, note that we can allow P_1 to perform arbitrary computations. Then it will follow from the privacy property of π that P_1 will be able to sample $\tilde{\omega}$ with all but negligible probability. P_1 then computes its first round message to P_3 assuming input 1 and internal randomness $\tilde{\omega}$. It then sends its first round messages to the parties, and accepts messages from them. In the second round, it does not send any messages and only accepts messages from other parties. Next, P_1 computes a value y_0' by invoking its output computation algorithm on input 0, internal randomness ω_1, round 1 messages received from all parties, and round 2 messages received from P_2 and P_4. Similarly, P_1 computes y_1' by invoking its output computation algorithm on input 1, internal randomness $\tilde{\omega}$, round 1 messages from all parties, and round 2 messages from P_3 and P_4. Finally, P_1 outputs the values y_0', y_1' as part of its view. We will show that with all but negligible probability it will hold that $y_0' = y_0$ and $y_1' = y_1$. Since an ideal-process adversary has access to P_4's input only via the trusted party implementing f, it is clear that it can obtain either y_0 or y_1 but not both. Thus, this suffices to establish the theorem. This is the high level idea; we now proceed to the formal details. Formally, P_1 does the following:

- Choose randomness ω_1 and compute $\tilde{m}_{1,2}^{(1)} \leftarrow A_{1,2}^{(1)}(0, \perp, \omega_1)$, and $\tilde{m}_{1,4}^{(1)} \leftarrow A_{1,4}^{(1)}(0, \perp, \omega_1)$.
- Choose random $\tilde{\omega}$ such that $A_{1,4}^{(1)}(1, \perp, \tilde{\omega}) = \tilde{m}_{1,4}^{(1)}$. If no such $\tilde{\omega}$ exists, output fail_1 and terminate.

- Compute $\tilde{m}_{1,3}^{(1)} \leftarrow A_{1,3}^{(1)}(1, \perp, \tilde{\omega})$.
- For $j = 2, 3, 4$, send message $\tilde{m}_{1,j}^{(1)}$ to P_j in round 1.
- Receive round 1 messages $m_{2,1}^{(1)}$, $m_{3,1}^{(1)}$, $m_{4,1}^{(1)}$, from other parties. Do not send any round 2 messages to any party. Receive round 2 messages $m_{2,1}^{(2)}$, $m_{3,1}^{(2)}$, $m_{4,1}^{(2)}$, from other parties and terminate the protocol.
- Compute and output $y_0' \leftarrow A_{1,1}^{(3)}(0, \{\{m_{k,i}^{(1)}\}_{k \in T_1}, \{m_{k,i}^{(2)}\}_{k \in \{2,4\}}\}; \omega_1)$, $y_1' \leftarrow A_{1,1}^{(3)}(1, \{\{m_{k,i}^{(1)}\}_{k \in T_1}, \{m_{k,i}^{(2)}\}_{k \in \{3,4\}}\}; \tilde{\omega})$.

First, we claim that corrupt P_1 does not output fail_1 with all but negligible probability, i.e., P_1 will be able to successfully find $\tilde{\omega}$ satisfying the conditions above. To show this, we rely on the privacy property of π against an (all-powerful) P_4. Clearly, if there exists no $\tilde{\omega}$ such that the output of $A_{1,4}^{(1)}$ on input 1 and internal randomness $\tilde{\omega}$, it is obvious to P_4 that P_1's input is 0, and thus privacy is violated. Therefore, it must hold with all but negligible probability (over the choice of ω) that such $\tilde{\omega}$ exists.

Next, we first assert that $y_0' = y_0$ holds with all but negligible probability. The key observation is that messages input to $A_{1,1}^{(3)}$ that are distributed identically to an execution where P_1 holds input 0 and a corrupt P_3 behaves honestly except it does not send its round 2 messages (i.e., aborts after round 1). Thus, it follows from the correctness of π that $y_0 = y_0'$ holds with all but negligible probability. Similarly, we assert that $y_1' = y_1$ holds with all but negligible probability. This is because the messages input to $A_{1,1}^{(3)}$ are distributed identically to an execution where P_1 holds input 1 and a corrupt party P_2 behaves honestly except it does not send its round 2 messages. Thus it follows from the correctness of π that $y_1' = y_1$ holds with all but negligible probability.

Finally we claim that no ideal-process adversary can generate a view with (y_0', y_1') such that these equal P_4's inputs with probability greater than 1/2. The key observation is that an ideal-process adversary has access to P_4's input only via the trusted party implementing f, it is clear that it can obtain either y_0 or y_1 but not both. In such a case, the best strategy for the ideal process adversary is to obtain one of them, and then simply try and guess the value of the other (thereby succeeding with probability 1/2). $\square$

It is instructive to note why the above impossibility does not apply to linear functions. Specifically for a linear function f, if the adversary P_1 can obtain an evaluation of f on input x_1 and honest inputs, then it can trivially obtain an evaluation of f on input $x_1' \neq x_1$ and the same honest inputs. Finally, we note that our negative result can be easily extended to hold in a setting with broadcast.

6 2-Round Computationally Secure 4-Party Computation

Protocol overview. For simplicity let us assume the existence of a broadcast channel. Our protocol proceeds by letting each party to broadcast a commitment of its input, and then CNF share the corresponding decommitment among the remaining parties. In the second round, parties execute pairwise PSMs that first attempts to reconstruct the inputs of all parties, and then compute the output from the reconstructed inputs. Unfortunately the general framework described as-is does not suffice for secure computation. For one, it may not always be possible to reconstruct input from shares distributed by a malicious party. Further, it may be the case that one pair of honest parties may hold consistent CNF shares from the malicious party while a different pair of honest parties may not. This is exacerbated by the fact that an honest party is guaranteed to receive output from only one PSM instance. In other words, even guaranteeing agreement on output seems somewhat nontrivial.

To circumvent the problems mentioned above, our protocol first detects whether the joint view of honest parties suffices to reconstruct the input of all parties. We do this by enhancing the PSM functionality in a way that lets parties ascertain if for every broadcasted commitment, there exists some pair of parties that hold (consistent) shares of the corresponding decommitment. (Indeed, this is our strategy for extracting the adversary's input in the simulation.) If a pair of parties do not hold consistent shares of a valid decommitment for some party's commitment, then the pairwise PSM in which the parties act as clients delivers as outputs the first round *views* of the honest clients. This in turn lets the referee to determine if its own shares coupled with shares from one of the clients suffices to reconstruct valid decommitments for all commitments. If this is indeed the case, then the referee can reconstruct all inputs from the joint views and then evaluate the function from scratch. On the other hand if there is some party whose commitment cannot be decommitted using the joint views, then the referee simply substitutes that party's input with 0, and evaluates the function from scratch using this new set of inputs. Of course, care must be taken not to reveal honest inputs to a malicious referee. We achieve this by letting the PSM check if the referee's commitment can be decommitted using shares held by honest clients, and then revealing the client views only if this check passes.

The ideas described above still do not suffice to address the somewhat subtler issue of agreement on output. We describe this issue in more detail below. Note that a malicious party that distributed shares of an invalid decommitment can ensure that all inputs are reconstructed successfully in *exactly one* of the PSM instances where it participated as a client and supplied shares of a valid decommitment. Thus, in this PSM instance the function will be evaluated on the reconstructed inputs. Note that this strategy lets exactly one honest party (that acted as referee in the PSM instance described above) to obtain directly the output of the function, while all other honest parties evaluate the function from scratch after substituting the malicious party's input with 0. In other words, the adversary can succeed in forcing different honest parties to obtain evaluations of the function on different sets of inputs. We use a somewhat counterintuitive idea to counter this adversarial strategy. Namely, we force the honest referee in the PSM instance to disregard the output of the function, and instead evaluate the function from scratch (using honest clients' views output in a different PSM instance) after substituting the malicious party's input with 0. To do this, we design the PSM functionality in a way that allows an honest referee to infer whether the joint view of the honest parties indeed contains a valid decommitments to all broadcasted commitments. In more detail, the PSM functionality will attempt to reconstruct the first round view of the referee from the views of the participating clients. (Note that this is possible due to the *efficient extendability* property of CNF sharing schemes.) Upon receiving this reconstructed view, the referee outputs the PSM output only if its view agrees with the reconstructed views. A formal description of the protocol appears in Appendix F. In Appendix F.3, we show how to remove the use of broadcast:

Theorem 6. *Assuming the existence of one-way permutations (alternatively, one-to-one one-way functions), there exists a 2-round 4-party computationally secure protocol over point-to-point channels for secure function evaluation that tolerates a single malicious party.*

References

1. Shashank Agrawal. Verifiable secret sharing in a total of three rounds. In *Info. Process. Lett. 112(22)*, pages 856–859, 2012.
2. Gilad Asharov, Abhishek Jain, Adriana Lopez-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *Eurocrypt*, pages 483–501, 2012.
3. A. Beimel. Secure schemes for secret sharing and key distribution. Ph.D. Thesis, Technion, 1996.
4. A. Beimel, Y. Ishai, R. Kumaresan, and E. Kushilevitz. On the cryptographic complexity of the worst functions. In *TCC*, pages 317–342, 2014.
5. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *Advances in Cryptology — Eurocrypt 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, 2011.
6. Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS 2008: 13th European Symposium on Research in Computer Security (ESORICS)*, volume 5283 of *LNCS*, pages 192–206. Springer, 2008.
7. Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In *Financial Cryptography and Data Security*, volume 5628 of *LNCS*, pages 325–343. Springer, 2009.
8. Seung Geol Choi, Ariel Elbaz, Tal Malkin, and Moti Yung. Secure multi-party computation minimizing online rounds. In *Advances in Cryptology — Asiacrypt 2009*, volume 5912 of *LNCS*, pages 268–286. Springer, December 2009.
9. S.G. Choi, J. Katz, A. Malozemoff, and V. Zikas. Efficient three-party computation from cut-and-choose. In *Crypto (2)*, pages 513–530, 2014.
10. I. Damgard, V. Pastro, N.P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Crypto*, pages 643–662, 2012.
11. Ivan Damgard and Sarah Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In *TCC*, pages 621–641, 2013.

12. Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 554–563. ACM Press, May 1994.
13. M.J. Fischer and N.A. Lynch. A lower bound for the time to assure interactive consiistency. In *Info. Process. Lett. 14(4)*, pages 183–186, 1982.
14. Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure mpc from indistinguishability obfuscation. In *TCC*, pages 74–94, 2014.
15. Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. The round complexity of verifiable secret sharing and secure multicast. In *33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 580–589. ACM Press, July 2001.
16. Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. On 2-round secure multiparty computation. In Moti Yung, editor, *Advances in Cryptology — Crypto 2002*, volume 2442 of *LNCS*, pages 178–193. Springer, 2002.
17. Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.
18. Shafi Goldwasser and Yehuda Lindell. Secure multi-party computation without agreement. *Journal of Cryptology*, 18(3):247–287, July 2005.
19. Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In *TCC*, pages 600–620, 2013.
20. Yuval Ishai, Eyal Kushilevitz, and Anat Paskin. Secure multiparty computation with minimal interaction. In *Advances in Cryptology — Crypto 2010*, volume 6223 of *LNCS*, pages 577–594. Springer, 2010.
21. Mitsuru Ito, Akira Saito, and Takao Nishizeki. Secret sharing schemes realizing general access structure. In *GLOBECOM*, pages 99–102, 1987.
22. Jonathan Katz and Chiu-Yuen Koo. Round-efficient secure computation in point-to-point networks. In Moni Naor, editor, *Advances in Cryptology — Eurocrypt 2007*, volume 4515 of *LNCS*, pages 311–328. Springer, 2007.
23. Jonathan Katz, Chiu-Yuen Koo, and Ranjit Kumaresan. Improving the round complexity of VSS in point-to-point networks. In *35th Intl. Colloquium on Automata, Languages, and Programming (ICALP), Part II*, volume 5126 of *LNCS*, pages 499–510. Springer, 2008.
24. Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In Matthew Franklin, editor, *Advances in Cryptology — Crypto 2004*, volume 3152 of *LNCS*, pages 335–354. Springer, 2004.
25. Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. In *TOPLAS 4(3)*, pages 382–401, 1982.
26. Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *Advances in Cryptology — Eurocrypt 2007*, volume 4515 of *LNCS*, pages 52–78. Springer, 2007.
27. M. Mahmoody and R. Pass. The curious case of non-interactive commitments - on the power of black-box vs. non-black-box use of primitives. In *Crypto*, pages 701–718, 2012.
28. A. Paskin-Cherniavsky. Secure computation with minimal interaction. Ph.D. Thesis, Technion, 2012.
29. Arpita Patra, Ashish Choudhary, Tal Rabin, and C. Pandu Rangan. The round complexity of verifiable secret sharing revisited. In Shai Halevi, editor, *Advances in Cryptology — Crypto 2009*, volume 5677 of *LNCS*, pages 487–504. Springer, 2009.

A Summary of Our Results

We summarize our results in Table 1. As noted there, none of our positive results require a broadcast channel, and our negative result holds even when parties have access to a broadcast channel.

n	Primitive	Security?	Output delivery?	Assump.?	Possible?	Broadcast?	Reference
3	SFE	stat.	selective abort	none	yes	no	Thm. 2
4	VSS	stat.	guaranteed	none	yes	no	Thm. 3
4	Linear FE	stat.	guaranteed	none	yes	no	Thm. 4
4	SFE	stat.	guaranteed	none	no	yes	Thm. 5
4	SFE	comp.	guaranteed	OWP/1-1 OWF	yes	no	Thm. 6
4	SFE/PP	stat.	guaranteed	none	yes	no	Thm. 7

Table 1. All results are for 2-round protocols. "SFE" stands for secure function evaluation, "Linear FE" stands for secure linear function evaluation, and "SFE/PP" stands for secure function evaluation in the preprocessing model.

B More Preliminaries and Related Work

Security for VSS. We give "simulation style" proofs for VSS, where we treat VSS essentially as a multi-receiver commitment scheme. Thus, when we simulate the view of a corrupt P_i this corresponds to proving privacy. When we simulate a corrupt D (and in particular extract its input), this corresponds to proving commitment. We chose to give simulation style proofs since it will become convenient to give intuition behind our design of protocols for secure function evaluation which as we will see build upon VSS protocols. In any case, our proofs can be trivially modified to give proofs of each VSS property separately.

Secure computation. We consider n-party protocols for $n = 3$ or 4, that involve two rounds of synchronous communication over secure point-to-point channels. All of our protocols are secure against rushing, adaptive adversaries, who may corrupt at most a single party. See [17] for more complete definitions. In addition to the standard simulation-based notions of full security (with guaranteed output delivery) and security with abort, we also consider *security with selective abort*. This notion, introduced in [18], differs from the standard notion of security with abort in that it allows the adversary (after learning its own outputs) to individually decide for each uncorrupted party whether this party will obtain its correct output or will output "$\perp$". Indeed, it was shown in [18] that two rounds of communication over point-to-point channels are sufficient to realize broadcast under this notion, with an arbitrary number of corrupted parties. Our notions of "security with abort" and "security with selective abort" correspond to the notions of "security with unanimous abort and no fairness" and "security with abort and no fairness" from [18]. To reiterate, security with selective abort is defined similarly to security with abort [17], except that the simulator can decide for each uncorrupted party whether this party will receive its output or $\perp$.

Secret sharing. An (n,t)-threshold secret sharing scheme, also referred to as a t-private secret sharing scheme, is an n-party secret sharing scheme in which every t parties learn nothing about the secret, and every $t+1$ parties can jointly reconstruct it. For formal definitions of secret sharing schemes see [3].

One property of secret sharing schemes that we exploit is *efficient extendability.* A secret sharing scheme is efficiently extendable, if for any subset $T \subseteq [n]$, it is possible to efficiently check whether the (purported) shares to T are consistent with a valid sharing of some secret s. Additionally, in case the shares are consistent, it is possible to efficiently sample a (full) sharing of some secret which is consistent with that partial sharing. This property is satisfied, in particular, by the schemes that we use, as well as any so-called "linear" secret sharing scheme. In this work, we rely on variants of standard linear secret sharing schemes, such as additive scheme and the CNF scheme which we describe below.

Additive sharing. Let G be any finite Abelian group. Additive sharing over G is the following k-out-of-k secret sharing scheme. To share $s \in G$, choose $k-1$ random elements $r_1, \dots, r_{k-1}$ from G, each element is chosen independently with uniform distribution. Compute $r_k = s - (\sum_{i=1}^{k-1} r_i)$. The share of P_i is r_i.

We will be mostly interested in 2-out-of-2 additive sharing over $\mathbb{F}_2$. Obviously, given both the shares r_1, r_2, we will be able to reconstruct the secret $s = r_1 \oplus r_2$. On the other hand, given the secret s and one of the shares r_1, we can determine the remaining share $r_2 = s \oplus r_1$.

CNF sharing [21]. Let G be any finite Abelian group. The t-private CNF sharing over G is a generalization of additive sharing where each party gets more than one group element. To share $s \in G$, additively share s into $\binom{k}{t}$ shares r_A, $A \in \binom{[k]}{t}$. The share of P_i consists of the $\binom{k-1}{t}$ group elements $\langle r_A \ : \ i \notin A \rangle$.

For each set $T \in \binom{[k]}{t}$, the parties in T do not get r_T and thus have no information about s. This implies that the scheme is t-private. We will be mostly interested in 1-private 3-party CNF sharing over $\mathbb{F}_2$; in this case we will write r_i instead of $r_{\{i\}}$. Obviously, given two of the three CNF shares, say $\langle r_1, r_2 \rangle, \langle r_2, r_3 \rangle$ we can reconstruct the secret $s = r_1 \oplus r_2 \oplus r_3$. On the other hand, given the secret s and one of the shares say $\langle r_1, r_2 \rangle$, we can determine the remaining shares; e.g., by first computing $r_3 = s \oplus r_1 \oplus r_2$, and setting the remaining shares as $\langle r_2, r_3 \rangle$ and $\langle r_3, r_1 \rangle$. When considering 1-private 3-party CNF sharing, we say that CNF shares held by P_i, i.e., $r^{(i)} = \langle r_A \ : \ i \notin A \rangle$ are "consistent" with CNF shares held by P_j, i.e., $r^{(j)} = \langle r'_A \ : \ j \notin A \rangle$ iff for every A such that $i, j \notin A$, it holds that $r_A = r'_A$, i.e., P_i and P_j agree on the additive shares held by both of them.

Non-interactive commitments. Our computationally secure 4-party protocol requires the use of non-interactive commitments which we define below. Note that such commitments can be constructed from one-way permutations (or even one-to-one one-way functions) (see [27] and references therein).

Definition 2. *A (non-interactive)* commitment scheme *for message space* $\{\mathcal{M}_\kappa\}$ *is a pair of* PPT *algorithms* $\mathsf{Com}, \mathsf{Dec}$ *such that for all* $\kappa \in \mathbb{N}$*, all messages* $m \in \mathcal{M}_\kappa$*, and all random coins* ω *it holds that* $\mathsf{Dec}(m, \mathsf{Com}(1^\kappa, m; \omega), \omega) = 1$*. A commitment scheme for message space* $\{\mathcal{M}_\kappa\}$ *is* secure *if it satisfies the following:*

Binding *For all* PPT *algorithms* $\mathcal{A}$ *the following is negligible in* κ*:*

$$\Pr\left[\begin{array}{l}(c,(m,\omega),(m',\omega')) \leftarrow \mathcal{A}(1^\kappa): \\ \qquad\qquad (m,\omega) \neq (m',\omega') \bigwedge \mathsf{Dec}(m,c,\omega)=1 \bigwedge \mathsf{Dec}(m',c,\omega')=1\end{array}\right]$$

Hiding *For all* PPT *algorithms* $\mathcal{A}$ *(that maintain state throughout their execution) the following is negligible in* κ*:*

$$\left|\Pr\left[\,(m_0,m_1) \leftarrow \mathcal{A}(1^\kappa);\, b \leftarrow \{0,1\}; c \leftarrow \mathsf{Com}(1^\kappa, m_b) : \mathcal{A}(c) = b\,\right] - \frac{1}{2}\right|.$$

$\diamondsuit$

Other related work. The round complexity of secure computation has been a subject of intense study. Constant-round 2-party protocols with security against malicious parties were given in several previous works; see e.g., [26] and references therein. In [24] it was shown that the optimal round complexity for secure 2-party computation without setup is 5 (where the negative result is restricted to protocols with black-box simulation). More relevant to our work is previous work on the round complexity of MPC with an honest majority and guaranteed output delivery. In this setting, constant-round protocols were given in several previous works; see e.g., [22] and references therein. In particular, it was shown in [15] that 3 rounds are sufficient for general secure computation with $t = \Omega(n)$ malicious parties, where one of the rounds requires broadcast. The question of minimizing the exact round complexity of MPC over point-to-point networks was explicitly considered in [22, 23], however the focus of these works was on obtaining nearly optimal resilience. Two-round protocols with guaranteed output delivery were given in [16] for specific functionalities. The round complexity of verifiable secret sharing (VSS) was initiated in [15] and subsequently studied in several works; see e.g., [1] and references therein. Finally, secure computation in the preprocessing model was studied in several previous works [5, 19, 10, 11, 4].

C More Details on 2-Round 3-Party Secure-with-Selective-Abort Protocol

C.1 Proof of Theorem 2

We first provide an informal overview of the simulator.

Overview. Denote the corrupt party by P_ℓ. Let P_i, P_j be the remaining (honest) parties. The simulator begins by sending random additive shares to the corrupt party on behalf of the honest parties. It also sends and receives randomness to be used in the PSM executions in the next round. Note that the simulator also receives additive shares from the corrupt party. Using the additive shares, the simulator computes the effective input say $\hat{x}_\ell$ of the corrupt party (i.e., by simply xor-ing the additive shares). Then, the simulator sends x_ℓ to the trusted party first, and obtains the output z_ℓ.

Next the simulator invokes the PSM simulator $\mathcal{S}^{\mathrm{trans}}_{\pi_{i,j}}$ (guaranteed by the privacy property) on inputs z_ℓ and the additive shares sent on behalf of the honest parties. Denote the output of the $\mathcal{S}^{\mathrm{trans}}_{\pi_{i,j}}$ by $\tau_{i,\ell}$ and $\tau_{j,\ell}$. Acting as the honest party P_i (resp. P_j), the simulator sends $\tau_{i,\ell}$ (resp. $\tau_{j,\ell}$) to the corrupt party. It remains to be shown how the simulator decides which uncorrupted parties learn the output and which receive $\perp$. To do this, the simulator does the following. First, acting as the honest party P_i the simulator receives the PSM message $\tau_{\ell,i}$ that P_ℓ sends to P_i as part of PSM execution $\pi_{\ell,j}$. Similarly, acting as P_j, the simulator also receives $\tau_{\ell,j}$. Next, the simulator invokes the PSM simulator $\mathcal{S}^{\mathrm{ext}}_{\pi_{\ell,i}}$ on the PSM message $\tau_{\ell,i}$ (and also the PSM randomness) to decide what effective input P_ℓ used in PSM subprotocol $\pi_{\ell,j}$. Depending on this input, the simulator then decides whether P_i will accept the output of $\pi_{\ell,j}$ or not. Specifically as in the real execution, the simulator checks if the shares input by P_ℓ are consistent with those held by P_i. If this is indeed the case, then the simulator asks the trusted party to deliver output to P_i, else it asks the trusted party to deliver $\perp$ to P_i. Whether P_j gets the output or not is also handled similarly by the simulator. This completes the sketch of the simulation.

We now formally describe the simulation for corrupt party, say P_ℓ below.

Simulating corrupt P_ℓ. For each $m \in T_\ell$, the simulator acting as P_m does the following:

- Choose random $x_{m,\ell}$ and send it to P_ℓ.
- Send PSM randomness $r^{\mathrm{psm}}_{m,\ell}$ to P_ℓ if $m < \ell$.
- Receive from P_ℓ values $x_{\ell,m}$.
- Receive from P_ℓ PSM randomness $r^{\mathrm{psm}}_{\ell,m}$ if $m > \ell$.

Next, the simulator extracts P_ℓ's input in a straightforward way:

Subroutine $\mathsf{Extract}_\ell(\{x_{\ell,m}\}_{m\in T_\ell})$

- Output $\hat{x}_\ell = \bigoplus_{m\in T_\ell} x_{\ell,m}$.

Next the simulator sends $\hat{x}_\ell$ to the trusted party. Let z_ℓ denote the output received from the trusted party. In the next step, the simulator prepares to send the second round messages to P_ℓ by executing the following for the pair (i,j) with $i < j$ and $\ell \notin \{i,j\}$.

Subroutine $\mathsf{PsmTrans}_\ell(x_{i,\ell}, x_{j,\ell}, x_{\ell,i}, x_{\ell,j}, z_\ell)$

- Set $\hat{z}_\ell = (z_\ell, x_{i,\ell}, x_{j,\ell}, x_{\ell,i}, x_{\ell,j})$.
- Invoke PSM simulator $\mathcal{S}^{\mathrm{trans}}_{\pi_{i,j}}(1^\sigma, \hat{z}_\ell)$ to obtain transcript $\tau_{i,\ell}, \tau_{j,\ell}$.
- For all $m \in \{i,j\}$ acting as P_m sends $\tau_{m,\ell}$ to P_ℓ over point-to-point channels.

For each $i \in T_\ell$, the simulator $\mathcal{S}$ receives PSM messages $\tilde{\tau}_{\ell,i}$ from the adversary for execution $\pi_{\ell,j}$ where $j \in [3] \setminus \{\ell, i\}$. (Recall that $r^{\text{psm}}_{\ell,j}$ denotes the PSM randomness used in execution $\pi_{\ell,j}$.) $\mathcal{S}$ then executes the following subroutine.

Subroutine $\mathsf{PsmExtract}_\ell(\{r^{\text{psm}}_{\ell,j}\}_{j\in T_\ell}, \{\tilde{\tau}_{\ell,i}\}_{i\in T_\ell}, (x_{i,\ell}, x_{j,\ell}, x_{\ell,i}, x_{\ell,j}))$

- For each $i \in T_\ell$: let $j \in [3] \setminus \{\ell, i\}$, and invoke PSM simulator $\mathcal{S}^{\text{ext}}_{\pi_{\ell,j}}(1^\sigma, r^{\text{psm}}_{\ell,j}, \tilde{\tau}_{\ell,i})$ to obtain output $\tilde{x}_{\ell,i}$.
- If $\exists i \in T_\ell$ such that $\tilde{x}_{\ell,i} = \perp$ (i.e., $\mathcal{S}^{\text{ext}}_{\pi_{\ell,j}}$ failed where $j \in [3] \setminus \{\ell, i\}$), then output psm-fail and terminate.
- Initialize $S_\ell = \emptyset$. For each $i \in T_\ell$ do:
  - Parse $\tilde{x}_{\ell,i} = (x'_{i,\ell}, x'_\ell, x'_{j,\ell})$.
  - If $x'_\ell \neq x_{\ell,i} \oplus x_{\ell,j}$ or $x'_{i,\ell} \neq x_{i,\ell}$ or $x'_{j,\ell} \neq x_{j,\ell}$, then add i to S_ℓ.
- Output S_ℓ.

If the output is psm-fail, then $\mathcal{S}$ outputs psm-fail and terminates. Else the simulator $\mathcal{S}$ sends (abort, S_ℓ) to the trusted party, outputs whatever the adversary outputs, and terminates the simulation.

Analysis. In order to show indistinguishability of the real and ideal executions, we first consider a hybrid experiment which is exactly the same as the real execution except that the PSM messages sent by the honest parties to P_ℓ are replaced by the simulated PSM transcripts generated by $\mathcal{S}^{\text{trans}}_{\pi_{i,j}}$. To generate these transcripts we first extract the input $\hat{x}_\ell$ by xor-ing the additive shares sent by P_ℓ, and then compute the output of $\pi_{i,j}$ using inputs provided by honest parties and $\hat{x}_\ell$. We then supply this output to $\mathcal{S}^{\text{trans}}_{\pi_{i,j}}$ to generate the simulated PSM transcripts. Now, we claim that the corrupt party's output in the hybrid execution is computed exactly as in the real execution. This follows from (1) the extracted input of the adversary $\hat{x}_\ell = x_{\ell,i} \oplus x_{\ell,j}$ equals the value x'_ℓ used by honest parties inside each PSM protocol that delivers output to P_ℓ, and (2) the correctness property of the PSM protocol in the real execution. Given this, it follows from the security (more precisely, the privacy property) of the PSM protocol that the joint distribution of the view of the adversary and honest outputs in the real protocol is indistinguishable from the corresponding distribution in the hybrid execution.

Next it is easy to see that the distribution of $\{x_{m,\ell}\}_{m\in T_\ell}$ is identical to the distribution in the real world, and further does not leak any information about the true inputs $\{x_m\}_{m\in T_\ell}$. It is also easy to see that the distribution of $\{r^{\text{psm}}_{m,\ell}\}_{m\,:\,0<m<\ell}$ in the ideal execution is identical to the same in the hybrid execution. Thus, we conclude that the view of the adversary in the hybrid execution is indistinguishable from the view of adversary in the ideal execution. Thus, to prove indistinguishability of the hybrid execution and the ideal execution it suffices to focus on the distribution of honest outputs. Note that in the ideal execution the honest outputs are generated using the true honest inputs and extracted input $\hat{x}_\ell$.

First, we claim that each honest party that accepts a non-$\perp$ output in the hybrid execution is ensured that this output is computed using the correct honest inputs and the corrupt input $\hat{x}_\ell$. To show the above, consider wlog how honest party P_i computes its output in the real execution from the output of the PSM execution $\pi_{\ell,j}$.

- Consider the corrupt input used to compute value z'_i inside $\pi_{\ell,j}$. Obviously if P_ℓ supplied input $\hat{x}_\ell = x_{\ell,i} \oplus x_{\ell,j}$ then this is used to compute z'_i. Else if P_ℓ supplied a different input $x'_\ell \neq \hat{x}_\ell$, then z'_i is computed using x'_ℓ but in this case, $x'_{\ell,i}$ computed inside the PSM satisfies $x'_{\ell,i} = x'_\ell \oplus x_{\ell,j} \neq \hat{x}_\ell \oplus x_{\ell,j} = x_{\ell,i}$ and therefore z'_i is not accepted by P_i.
- Obviously honest P_j supplies the correct input x_j and this is used to compute z'_i in $\pi_{\ell,j}$.
- The input of P_i is first reconstructed using shares provided by P_ℓ and P_j. Obviously honest P_j supplies the correct share $x_{i,j}$. If P_ℓ also supplies the correct share, then the value z'_i is computed using the correct input of P_i, i.e., x_i. On the other hand, if P_ℓ supplied an incorrect share say $x'_{i,\ell}$, then the key observation is that this value $x'_{i,\ell}$ will be revealed to P_i by $\pi_{\ell,j}$, and thus P_i will not accept this z'_i value.

Given the above it remains to be shown that the set of honest parties that receive $\perp$ in the ideal execution equals the set of honest parties that output $\perp$ in the real execution. To prove the above, we use the fact that for all $j \in T_\ell$, with all but negligible probability the PSM simulator $\mathcal{S}^{\mathrm{ext}}_{\pi_{\ell,j}}$ extracts the input supplied by P_ℓ in PSM execution $\pi_{\ell,j}$. The above follows from the robustness property of the PSM protocol (which guarantees the existence of such a $\mathcal{S}^{\mathrm{ext}}_{\pi_{\ell,j}}$), and in particular, we have that $\mathcal{S}$ outputs $\mathsf{psm\text{-}fail}$ with negligible probability. It then follows by simple inspection that the criterion used to add i to S_ℓ in the simulation is essentially the same as the criterion used by P_i to reject the output z'_i of the PSM protocol $\pi_{\ell,j}$ in the hybrid execution. With this we conclude that the joint distribution of the view of the adversary and the outputs of the honest parties in the ideal execution is indistinguishable from the joint distribution of the view of the adversary and the outputs of the honest parties in the hybrid execution. This completes the analysis of the simulation.

D More Details on 2-Round 4-Party Statistical VSS

D.1 Analysis of the Naïve Protocol

Analysis. Clearly the protocol satisfies the privacy requirement. Since D does not send messages after the sharing phase, the commitment property also holds. Next we show that except under specific adversary strategies, the protocol also satisfies correctness.

Claim. Unless G contains exactly one edge, the protocol described is correct.
Proof. Let $i, j, k \in [3]$ be distinct indices. Wlog, let P_i, P_j be honest parties. Note that if D is honest, then $(i,j) \notin G$, and it holds that $s = \mathsf{rec}_{i,j} = \mathsf{rec}_{j,i}$. Now if $\exists k$ such that $(i,k) \notin G$, then since G does not contain exactly one edge, it must hold that $(j,k) \notin G$, and therefore $\mathsf{rec}_{i,k} = \mathsf{rec}_{j,k} = \mathsf{rec}_{i,j} = s$ (i.e., all parties broadcasted consistent CNF shares). Thus all honest parties reconstruct

s. On the other hand if for $k \in T_{i,j}$ it holds that $(i,k) \in G$ and $(j,k) \in G$, then honest parties reconstruct $s = \mathsf{rec}_{i,j} = \mathsf{rec}_{j,i}$ (since $(i,j) \notin G$). Therefore we have shown that if D is honest, then honest parties reconstruct D's input s. □

D.2 2-Round 4-Party VSS with a Broadcast Channel

Protocol description. We show how to modify the protocol from our first attempt to solve the problem of 2-round 4-party VSS in Figure 2. The modified steps are highlighted with a "$\star$" symbol next to them.

Sharing Phase. The dealer CNF shares its secret s among the remaining parties. That is, it chooses random s_1, s_2, s_3 subject to $\bigoplus_{i=1,2,3} s_i = s$, and sends CNF share $\{s_j\}_{j \neq i}$ to party P_i for $i \in [3]$.

- $\star$ D also creates σ information-theoretic MACs for each share s_j as $\{M_{j,\ell}^{(i)}, K_{j,\ell}^{(i)}\}_{i \neq j, \ell \in [\sigma]}$, and sends $\{M_{j,\ell}^{(i)}\}_{\ell \in [\sigma]}$ to P_i for each $i \neq j$, and $\{K_{j,\ell}^{(i)}\}_{i \neq j, \ell \in [\sigma]}$ to P_j.

Reconstruction Phase.

- $\star$ Each P_j sends $(S_{j,i}, \{K_{j,\ell}^{(i)}\}_{\ell \in S_{j,i}})$ to P_i for every $i \neq j$, for randomly chosen $S_{j,i} \subset [\sigma]$ of size $\sigma/2$, and $(S_{j,i}, \{K_{j,\ell}^{(i)}\}_{\ell \in [\sigma]})$ to P_k for $k \neq i$.
- $\star$ Each P_i broadcasts $\{s_j^{(i)} = s_j\}_{j \neq i}$ and $\{M_{j,\ell}^{(i)}\}_{j \neq i, \ell \in [\sigma]}$.

Local Computation. D outputs s and terminates the protocol. For every $j, k \in [3]$, define $\mathsf{rec}_{j,k} = s_j^{(k)} \oplus \bigoplus_{i \neq j} s_i^{(j)}$. For each $m \in [3]$, party P_m reconstructs output as follows:

- Let G denote the 3-vertex inconsistency graph such that $(i,j) \in G$ iff $\exists k \in [3] \setminus \{i,j\}$ such that $s_k^{(i)} \neq s_k^{(j)}$.
- If G contains exactly one edge, say (i,j) with $k \in [3] \setminus \{i,j\}$ such that $s_k^{(i)} \neq s_k^{(j)}$, then
  - for every $m' \in \{i,j\}$, initialize $c_{m'}^{(m)} = 0$.
  - if $m \in \{i,j\}$: set $c_m^{(m)} = 1$ if $\forall \ell \in S_{k,m}$ it holds that $M_{k,\ell}^{(m)}$ is a MAC on $s_k^{(m)}$ consistent with $K_{k,\ell}^{(m)}$.
  - for $m' \in \{i,j\} \setminus \{m\}$, set $c_{m'}^{(m)} = 1$ if (1) $\forall \ell \in S_{k,m'}$ it holds that $M_{k,\ell}^{(m')}$ is a MAC on $s_k^{(m')}$ consistent with key $K_{k,\ell}^{(m')}$, and (2) $\exists \ell \in [\sigma] \setminus S_{k,m'}$ such that $M_{k,\ell}^{(m')}$ is a MAC on $s_k^{(m')}$ consistent with key $K_{k,\ell}^{(m')}$.
  - P_m outputs $\perp$ if $c_i^{(m)} = c_j^{(m)}$, else outputs $\mathsf{rec}_{i,k}$ if $c_i^{(m)} = 1$, else outputs $\mathsf{rec}_{j,k}$ if $c_j^{(m)} = 1$.
- Else if $\exists (j,k) \notin G$, then party P_m outputs $\mathsf{rec}_{j,k}$. If there is no such j,k, then P_m outputs $\perp$.

Fig. 2. 4-party statistical VSS protocol with 1-round sharing and 1-round reconstruction.

In Appendix D.3, we prove the following lemma.

Lemma 1. *There exists a 4-party statistically secure protocol for verifiable secret sharing that tolerates a single malicious party and requires one round in the sharing phase and one round (which includes use of broadcast channel) in the reconstruction phase.*

D.3 Proof of Lemma 1

We split the analysis depending on whether D is corrupt or not.

Simulating a corrupt D. The simulator obtains $\{s_j^{(i)}\}_{j\neq i}$ for each $i \in [3]$. Then, for each share s_j, it obtains $\{M_{j,\ell}^{(i)}\}_{\ell\in[\sigma]}$ acting as P_i for each $i \neq j$, and $\{K_{j,\ell}^{(i)}\}_{i\neq j,\ell\in[\sigma]}$ acting as P_j. It then constructs a 3-vertex inconsistency graph G' which contains an edge between vertices $i, j \in [3]$ iff $\exists k \in [3] \setminus \{i, j\}$ such that $s_k^{(i)} \neq s_k^{(j)}$. It then extracts the dealer input as follows:

- If G' contains exactly one edge, say (i, j), then for each $m \in \{i, j\}$, initialize $c_m = 0$, then pick random $S_m \subset [\sigma]$ of size $\sigma/2$, and set $c_m = 1$ if (1) $\forall \ell \in S_m$ it holds that $M_{k,\ell}^{(m)}$ is a MAC on $s_k^{(m)}$ that is consistent with key $K_{k,\ell}^{(m)}$, and (2) $\exists \ell \in [\sigma] \setminus S_m$ such that $M_{k,\ell}^{(m)}$ is a MAC on $s_k^{(m)}$ that is consistent with key $K_{k,\ell}^{(m)}$. If $c_i = c_j$, send $\perp$ to the trusted party, else send $\mathsf{rec}_{m,k}$ to the trusted party where $m \in \{i, j\}$ such that $c_m = 1$.
- Else, if there exists $(j, k) \notin G'$, then it sends $\mathsf{rec}_{j,k}$ to the trusted party.
- Else, it sends $\perp$ to the trusted party.

Then, it simulates the reconstruction phase of the protocol by sending messages as computed by honest P_1, P_2, P_3. Finally, it outputs whatever the adversary outputs, and terminates.

Analysis. First, consider the case when G' contains exactly one edge, say (i, j). Let $k \in [3] \setminus \{i, j\}$. Observe that the view of the adversary in the real execution is indistinguishable from its view in the ideal execution. On the other hand, we will show that the output of the honest parties in the ideal execution and the real execution differ only with probability negligible in σ. First, observe that the distribution of S_i, S_j in the simulation is identical to the distribution of $S_{k,i}, S_{k,j}$ in the real execution. Given this, it follows that the distribution of c_i (resp. c_j) is identical to the distribution of $c_i^{(k)}$ as well as $c_i^{(j)}$ (resp. $c_j^{(k)}$ as well as $c_j^{(i)}$). Note in particular that $c_i^{(k)} = c_i^{(j)}$, and that $c_j^{(k)} = c_j^{(i)}$. Next, we claim that the distribution of c_i (resp. c_j) is statistically indistinguishable from the distribution of $c_i^{(i)}$ (resp. $c_j^{(j)}$). This is because the two distributions differ only if for some $m \in \{i, j\}$, (1) $\forall \ell \in S_m$ it holds that $M_{k,\ell}^{(m)}$ is a MAC on $s_k^{(m)}$ that is consistent with $K_{k,\ell}^{(m)}$, and (2) $\forall \ell \in [\sigma] \setminus S_m$ it holds that $M_{k,\ell}^{(m)}$ is *not* a MAC on $s_k^{(m)}$ that is consistent with key $K_{k,\ell}^{(m)}$. It is easy to see that this event happens with probability $\mathsf{negl}(\sigma)$ over random choice of S_m in the ideal execution. Thus, conditioned on this event not happening, we have that the distribution of c_i (resp. c_j) is identical to the distribution of $c_i^{(i)} = c_i^{(j)} = c_i^{(k)}$ (resp. $c_j^{(i)} = c_j^{(j)} = c_j^{(k)}$).

Given this, it follows that distribution of (honest) outputs in the real and ideal executions are statistically indistinguishable.

Next, consider the case when G' does not contain exactly one edge. As before it is easy to see that the view of the adversary in the real execution is distributed identically to its view in the ideal execution. We claim that the outputs of the honest parties in the real execution and the ideal execution are distributed identically. This is because (1) when there exists unique $(j,k) \notin G'$ then $\mathsf{rec}_{j,k} = \mathsf{rec}_{k,j}$ holds and all parties output $\mathsf{rec}_{j,k}$, and (2) when G' contains all three edges, then all parties output $\perp$, and (3) when G' contains no edges, then for every unique $i,j,k \in [3]$, it holds that $\mathsf{rec}_{i,j} = \mathsf{rec}_{j,k} = \mathsf{rec}_{i,k}$, and thus all parties output $\mathsf{rec}_{j,k}$ for some distinct $j,k \in [3]$. In all cases, it is easy to see that the simulation is perfectly indistinguishable from the real execution.

Simulating a corrupt P_i. Acting as D, the simulator sends random shares $\{s_j\}_{j\neq i}$ to P_i. Then for $j \neq i$, it samples random (but consistent) $\{M^{(i)}_{j,\ell}, K^{(i)}_{j,\ell}\}_{j\neq i,\ell\in[\sigma]}$ on value s_j. In addition it samples a random set of keys (for the unknown share s_i) $\{\widetilde{K}^{(j)}_{i,\ell}\}_{j\neq i,\ell\in\sigma}$. It then sends $\{M^{(i)}_{j,\ell}\}_{\ell\in[\sigma],j\neq i}$ and $\{\widetilde{K}^{(j)}_{i,\ell}\}_{j\neq i,\ell\in[\sigma]}$ to P_i. At the beginning of the reconstruction phase, the simulator receives secret s from the trusted party. It then sets $s_i = s \oplus \bigoplus_{j\neq i} s_j$, and creates MACs on s_i, say $\{\widetilde{M}^{(j)}_{i,\ell}\}_{j\neq i,\ell\in[\sigma]}$ that are consistent with keys $\{\widetilde{K}^{(j)}_{i,\ell}\}_{j\neq i,\ell\in[\sigma]}$. For each $j \in [3] \setminus \{i\}$, the simulator acting as P_j sends the following to (corrupt) P_i:

- values $\{s_m\}_{m\neq j}$, $(S_{j,i}, \{K^{(i)}_{j,\ell}\}_{\ell\in S_{j,i}})$, $(S_{j,k}, \{K^{(k)}_{j,\ell}\}_{\ell\in[\sigma]})$ for $k \in [3] \setminus \{i,j\}$ and randomly chosen $S_{j,i}, S_{j,k} \subset [\sigma]$ each of size $\sigma/2$ over the point-to-point channel.
- values $\{\widetilde{M}^{(j)}_{i,\ell}\}_{\ell\in[\sigma]}$ and $\{\widetilde{M}^{(j)}_{k,\ell}\}_{\ell\in[\sigma]}$ where $k \in [3] \setminus \{i,j\}$ over the broadcast channel.

(Throughout the protocol, the simulator ignores values sent by P_i.) Finally, it outputs whatever the adversary outputs, and terminates.

Analysis. First, note that the view of the adversary in the real execution is indistinguishable from its view in the ideal execution. Therefore, the simulation is indistinguishable as long as the honest parties output the dealer's input s in the real execution. Let j,k be distinct indices in $[3] \setminus \{i\}$. Note that when D is honest, all edges in G' involve i since honest P_j, P_k agree on their common share s_i. Thus, $(j,k) \notin G'$ and further, $\mathsf{rec}_{j,k} = \mathsf{rec}_{k,j} = s$ also holds. Clearly, when G' contains two edges (i.e., both involving i), then correctness holds since all parties reconstruct $\mathsf{rec}_{j,k}$. Next, when G' contains no edges, this means that for all $m' \in \{j,k\}$, it holds that $\mathsf{rec}_{i,m'} = \mathsf{rec}_{m',i} = \mathsf{rec}_{j,k}$, and once again correctness holds since all parties reconstruct $\mathsf{rec}_{j,k}$.

The remaining case is when G contains a single edge, say (i,j). Note that $c^{(j)}_i$ always equals $c^{(k)}_i$ for honest P_j, P_k. Then, it is easy to see that correctness holds as long as $c^{(j)}_i = c^{(k)}_i = 0$, and in particular, parties reconstruct $\mathsf{rec}_{j,k}$. On the other hand, if for some $m \in \{j,k\}$ such that $c^{(m)}_i = 1$, then correctness does not hold. It remains to be shown that this event happens with negligible probability.

Indeed such an event happens only if corrupt P_i can produce some $M^{(i)}_{k,\ell}$ such that $\ell \notin S_{k,i}$, and $M^{(i)}_{k,\ell}$ is a MAC on $s^{(i)}_k \neq s_k$ that is consistent with the key $K^{(i)}_{k,\ell}$. Note that the values $\{K^{(i)}_{k,\ell}\}_{\ell \notin S_{k,i}}$ are completely hidden from P_i since it is generated by honest D, and sent to P_k via point-to-point channels, which P_k then sends it to P_j again via point-to-point channels. Since corrupt P_i can forge this MAC only with probability $\mathsf{negl}(\sigma)$, it follows that correctness holds with all but negligible probability. Therefore, we conclude that the simulation is statistically indistinguishable from the real execution.

D.4 2-Round 4-Party Statistical VSS Protocol Over Point-to-Point Channels

We now show how to remove the use of broadcast channel from our VSS protocol in Section 4. We provide an overview of our protocol.

Protocol overview. Note that in our VSS construction in Section 4 (henceforth referred to as the "original protocol"), the broadcast channel was used only in the reconstruction phase. Our idea to remove the use of broadcast channel in the reconstruction phase is simple: we just let parties transmit their broadcast values over each point-to-point channel. It is easy to see that the round complexity of the protocol (as well as its privacy) is preserved. The non-triviality is in showing that the resulting protocol is still a VSS protocol. The main challenge is that now parties do not hold the same inconsistency graph.

Suppose that D is honest. Assume wlog that P_i is corrupt. Let $j, k \in [3] \setminus \{i\}$ be distinct indices. As pointed out above, we cannot assume that parties P_j and P_k hold the same inconsistency graph. The only thing we are guaranteed is that (j,k) will not be an edge in inconsistency graphs G_j (resp. G_k) held by P_j (resp. P_k). Thus our goal will be to anchor the parties' decision to output $\mathsf{rec}_{j,k} = \mathsf{rec}_{k,j} = s$. We will focus on the local view of P_j. (The case involving P_k is handled similarly.) When G_j contains no edge, it must hold that values $s^{(i)}_j, s^{(i)}_k$ received from P_i must equal the true values s_j, s_k, and therefore for any $m \in \{j,k\}$ it must hold that $\mathsf{rec}_{i,m} = \mathsf{rec}_{m,i} = \mathsf{rec}_{j,k} = s$. Next, if G_j contains two edges (i.e., (i,j) and (i,k)), then P_j will output $\mathsf{rec}_{j,k}$ as in the original protocol, and therefore correctness holds. Finally suppose there is exactly one edge in G_j. If the edge is (i,j), then as in the analysis of the original protocol, (1) P_j will conclude that P_k is honest, (2) P_j's reveal will be accepted by P_j itself, and (3) with all but negligible probability, P_i's reveal will be rejected by P_j. On the other hand, if the edge is (i,k), then (1) P_k's reveal will be accepted by P_j, and (3) with all but negligible probability, P_i's reveal will be rejected by P_j. Thus in either case, P_j outputs $\mathsf{rec}_{j,k}$ as in the original protocol, and thus correctness holds.

Next consider the case when D is corrupt. Observe that (1) the original protocol uses the broadcast channel only in the reconstruction phase, and (2) D does not act during the reconstruction phase of the protocol. Next, note that when the sender of a broadcast is honest, the broadcast channel can be safely replaced by use of point-to-point channels. This combined with the two observations above immediately provides intuition as to why we essentially obtain the same guarantees as the original protocol (i.e., which uses a broadcast channel) when D is corrupt.

We now proceed to the formal protocol description. The modified steps are highlighted with a "$\star$" symbol next to them.

Sharing Phase. The dealer CNF shares its secret s among the remaining parties. More precisely, it chooses random s_1, s_2, s_3 subject to $\bigoplus_{i=1,2,3} s_i = s$, and sends CNF share $\{s_j\}_{j \neq i}$ to party P_i for $i \in [3]$. D also creates σ information-theoretic MACs for each share s_j as $\{M_{j,\ell}^{(i)}, K_{j,\ell}^{(i)}\}_{i \neq j, \ell \in [\sigma]}$, and sends $\{M_{j,\ell}^{(i)}\}_{\ell \in [\sigma]}$ to P_i for each $i \neq j$, and $\{K_{j,\ell}^{(i)}\}_{i \neq j, \ell \in [\sigma]}$ to P_j.

Reconstruction Phase. Each P_j sends $(S_{j,i}, \{K_{j,\ell}^{(i)}\}_{\ell \in S_{j,i}})$ to P_i for every $i \neq j$, for randomly chosen $S_{j,i} \subset [\sigma]$ of size $\sigma/2$, and $(S_{j,i}, \{K_{j,\ell}^{(i)}\}_{\ell \in [\sigma]})$ to P_k for $k \neq i$.

- $\star$ Each party P_i sends $\{s_j^{(i)}\}_{j \neq i}$, $\{M_{j,\ell}^{(i)}\}_{j \neq i, \ell \in [\sigma]}$ over point-to-point channels to each P_k for $k \neq i$. Let P_k receive these values as $\{s_j^{(i,k)}\}_{j \neq i}$, $\{M_{j,\ell}^{(i,k)}\}_{j \neq i, \ell \in [\sigma]}$.

$\star$ *Local Computation.* D outputs s and terminates the protocol. For every (possibly non-distinct) $i, j, k \in [3]$, define $\mathsf{rec}_{j,k}^{(i)} = s_j^{(k,i)} \oplus \bigoplus_{i \neq j} s_i^{(j,i)}$. For each $m \in [3]$, party P_m reconstructs output as follows:

- Let G_m denote the 3-vertex inconsistency graph which contains an edge between vertices $i, j \in [3]$ iff $\exists k \in [3] \setminus \{i, j\}$ such that $s_k^{(i,m)} \neq s_k^{(j,m)}$.
- If G_m contains exactly one edge, say (i, j) with $k \in [3] \setminus \{i, j\}$ such that $s_k^{(i,m)} \neq s_k^{(j,m)}$, then
  - for every $m' \in \{i, j\}$, party P_m initializes $c_{m'}^{(m)} = 0$.
  - if $m \in \{i, j\}$, then party P_m sets $c_m^{(m)} = 1$ if $\forall \ell \in S_{k,m}$ it holds that $M_{k,\ell}^{(m)}$ is a MAC on $s_k^{(m)}$ consistent with key $K_{k,\ell}^{(m)}$.
  - for $m' \in \{i, j\} \setminus \{m\}$, party P_m sets $c_{m'}^{(m)} = 1$ if (1) $\forall \ell \in S_{k,m'}$ it holds that $M_{k,\ell}^{(m',m)}$ is a MAC on $s_k^{(m',m)}$ consistent with key $K_{k,\ell}^{(m')}$, and (2) $\exists \ell \in [\sigma] \setminus S_{k,m'}$ such that $M_{k,\ell}^{(m',m)}$ is a MAC on $s_k^{(m',m)}$ consistent with key $K_{k,\ell}^{(m')}$.
  - P_m outputs $\perp$ if $c_i^{(m)} = c_j^{(m)}$, else outputs $\mathsf{rec}_{i,k}^{(m)}$ if $c_i^{(m)} = 1$, else outputs $\mathsf{rec}_{j,k}^{(m)}$ if $c_j^{(m)} = 1$.
- Else if $\exists (j, k) \notin G_m$, then party P_m outputs $\mathsf{rec}_{j,k}^{(m)}$. If there is no such j, k, then P_m outputs $\perp$.

Theorem 3. (restated) There exists a 4-party statistically secure protocol for verifiable secret sharing over point-to-point channels that tolerates a single malicious party and requires one round in the sharing phase and one round in the reconstruction phase.

Proof. We split the analysis depending on whether D is corrupt or not.

Simulating a corrupt D. The simulator obtains $\{s_j^{(i)}\}_{j\neq i}$ for each $i \in [3]$. Then, for each share s_j, it obtains $\{M_{j,\ell}^{(i)}\}_{\ell\in[\sigma]}$ acting as P_i for each $i \neq j$, and $\{K_{j,\ell}^{(i)}\}_{i\neq j,\ell\in[\sigma]}$ acting as P_j. It then constructs a 3-vertex inconsistency graph G' which contains an edge between vertices $i, j \in [3]$ iff $\exists k \in [3]$ such that $s_k^{(i)} \neq s_k^{(j)}$. It then extracts the dealer input as follows:

- If G' contains exactly one edge, say (i, j), then for each $m \in \{i, j\}$, initialize $c_m = 0$, then pick random $S_m \subset [\sigma]$ of size $\sigma/2$, and set $c_m = 1$ if (1) $\forall \ell \in S_m$ it holds that $M_{k,\ell}^{(m)}$ is a MAC on $s_k^{(m)}$ that is consistent with key $K_{k,\ell}^{(m)}$, and (2) $\exists \ell \in [\sigma] \setminus S_m$ such that $M_{k,\ell}^{(m)}$ is a MAC on $s_k^{(m)}$ that is consistent with key $K_{k,\ell}^{(m)}$. If $c_i = c_j$, send $\perp$ to the trusted party, else send $\mathsf{rec}_{m,k}$ to the trusted party where $m \in \{i, j\}$ such that $c_m = 1$.
- Else, if there exists $(i, j) \notin G'$, then it sends $\mathsf{rec}_{j,k}$ to the trusted party.
- Else, it sends $\perp$ to the trusted party.

Then, it simulates the reconstruction phase of the protocol by sending messages as computed by honest P_1, P_2, P_3. Finally, it outputs whatever the adversary outputs, and terminates.

Analysis. Note that the description of the simulator is exactly the same as in the case when we were allowed use of broadcast channel. This is because in the previous construction only parties other than D used the broadcast channel. Therefore, when D is corrupt, the parties whose broadcast was replaced by transmissions point-to-point channels were all honest. That is, in this case, there is absolutely no difference between the use of broadcast channels or the use of point-to-point channels. In particular, all honest parties P_1, P_2, P_3 hold the *same inconsistency graph*, say G'. Furthermore since all parties P_1, P_2, P_3 are honest, we have that for all $m, m', m'' \in [3]$, it holds that $s_m^{(m',m'')} = s_m^{(m')}$. Consequently, we also have that for all $m, m', m'' \in [3]$, it holds that $\mathsf{rec}_{m',m''}^{(m)} = \mathsf{rec}_{m',m''}$. Thus, in this case, the analysis of the simulation is the same as that of the original protocol. For the sake of completeness, we describe the analysis below.

First, consider the case when G' contains exactly one edge, say (i, j). (As we will see below, this case is handled exactly as in the original protocol. This is because in this case, decisions made by the parties in the original protocol are based on values received over point-to-point channels.) Let $k \in [3] \setminus \{i, j\}$. Observe that the view of the adversary in the real execution is indistinguishable from its view in the ideal execution. On the other hand, we will show that the output of the honest parties in the ideal execution and the real execution differ only with probability negligible in σ. First, observe that the distribution of S_i, S_j in the simulation is identical to the distribution of $S_{k,i}, S_{k,j}$ in the real execution. Given this, it follows that the distribution of c_i (resp. c_j) is identical to the distribution of $c_i^{(k)}$ as well as $c_i^{(j)}$ (resp. $c_j^{(k)}$ as well as $c_j^{(i)}$). Note in particular that $c_i^{(k)} = c_i^{(j)}$, and that $c_j^{(k)} = c_j^{(i)}$. Next, we claim that the distribution of

c_i (resp. c_j) is statistically indistinguishable from the distribution of $c_i^{(i)}$ (resp. $c_j^{(j)}$). This is because the two distributions differ only if for some $m \in \{i,j\}$, (1) $\forall \ell \in S_m$ it holds that $M_{k,\ell}^{(m)}$ is a MAC on $s_k^{(m)}$ that is consistent with $K_{k,\ell}^{(m)}$, and (2) $\forall \ell \in [\sigma] \setminus S_m$ it holds that $M_{k,\ell}^{(m)}$ is *not* a MAC on $s_k^{(m)}$ that is consistent with key $K_{k,\ell}^{(m)}$. It is easy to see that this event happens with probability $\mathsf{negl}(\sigma)$ over random choice of S_m in the ideal execution. Thus, conditioned on this event not happening, we have that the distribution of c_i (resp. c_j) is identical to the distribution of $c_i^{(i)} = c_i^{(j)} = c_i^{(k)}$ (resp. $c_j^{(i)} = c_j^{(j)} = c_j^{(k)}$). Given this, it follows that distribution of (honest) outputs in the real and ideal executions are statistically indistinguishable.

Next, consider the case when G' does not contain exactly one edge. As before it is easy to see that the view of the adversary in the real execution is distributed identically to its view in the ideal execution. We claim that the outputs of the honest parties in the real execution and the ideal execution are distributed identically. This is because (1) when there exists unique $(j,k) \notin G'$ then $\mathsf{rec}_{j,k} = \mathsf{rec}_{k,j}$ holds and all parties output $\mathsf{rec}_{j,k}$, and (2) when G' contains all three edges, then all parties output $\perp$, and (3) when G' contains no edges, then for every unique $i,j,k \in [3]$, it holds that $\mathsf{rec}_{i,j} = \mathsf{rec}_{j,k} = \mathsf{rec}_{i,k}$, and thus all parties output $\mathsf{rec}_{j,k}$ for some distinct $j,k \in [3]$. In all cases, it is easy to see that the simulation is perfectly indistinguishable from the real execution.

Simulating a corrupt P_i. Acting as D, the simulator sends random shares $\{s_j\}_{j \neq i}$ to P_i. Then for $j \neq i$, it samples random (but consistent) $\{M_{j,\ell}^{(i)}, K_{j,\ell}^{(i)}\}_{j \neq i, \ell \in [\sigma]}$ on value s_j. In addition it samples a random set of keys (for the unknown share s_i) $\{\widetilde{K}_{i,\ell}^{(j)}\}_{j \neq i, \ell \in \sigma}$. It then sends $\{M_{j,\ell}^{(i)}\}_{\ell \in [\sigma], j \neq i}$ and $\{\widetilde{K}_{i,\ell}^{(j)}\}_{j \neq i, \ell \in [\sigma]}$ to P_i. At the beginning of the reconstruction phase, the simulator receives secret s from the trusted party. It then sets $s_i = s \oplus \bigoplus_{j \neq i} s_j$, and creates MACs on s_i, say $\{\widetilde{M}_{i,\ell}^{(j)}\}_{j \neq i, \ell \in [\sigma]}$ that are consistent with keys $\{\widetilde{K}_{i,\ell}^{(j)}\}_{j \neq i, \ell \in [\sigma]}$. For each $j \in [3] \setminus \{i\}$, the simulator acting as P_j sends the following to (corrupt) P_i:

- values $\{s_m\}_{m \neq j}$, $(S_{j,i}, \{K_{j,\ell}^{(i)}\}_{\ell \in S_{j,i}})$, $(S_{j,k}, \{K_{j,\ell}^{(k)}\}_{\ell \in [\sigma]})$ for $k \in [3] \setminus \{i,j\}$ and randomly chosen $S_{j,i}, S_{j,k} \subset [\sigma]$ each of size $\sigma/2$ over the point-to-point channel.
- values $\{\widetilde{M}_{i,\ell}^{(j)}\}_{\ell \in [\sigma]}$ and $\{\widetilde{M}_{k,\ell}^{(j)}\}_{\ell \in [\sigma]}$ where $k \in [3] \setminus \{i,j\}$ over the point-to-point channel.

(Throughout the protocol, the simulator ignores values sent by P_i.) Finally, it outputs whatever the adversary outputs, and terminates.

Analysis. Note that the description of the simulator is almost identical to the case when we were allowed use of broadcast channel. The only difference is that in the reconstruction phase, the simulator has to send the MAC values through point-to-point channels (instead of the broadcast channel). It is easy to see that the view of the adversary in the ideal execution is indistinguishable from its view in the real execution. The analysis below shows that with overwhelming probability, output of honest parties in the real execution is the same as in the ideal execution, i.e., it equals the input of the dealer.

Let $j, k \in [3] \setminus \{i\}$ be distinct indices. Note that P_j and P_k are honest. We prove that P_j outputs s with high probability. (The argument is identical for P_k.) Let G_j be the inconsistency graph in the view of P_j. Note that since D is honest, honest parties have consistent shares, and therefore, $(j,k) \notin G_j$ (and in particular, G_j has less than three edges). We first analyze the case when G_j is empty. In this case, (corrupt) P_i's shares received by P_j must equal the shares sent to P_i by D, else either P_j or P_k will have a conflict with P_i in G_j. Therefore, in this case, P_j reconstructs $\mathsf{rec}^{(j)}_{j,k} = s$. Next, we consider the case when G_j has 2 edges. Since we have $(j,k) \notin G_j$, the edges must be (i,j) and (i,k). Therefore, P_j reconstructs $\mathsf{rec}^{(j)}_{j,k} = s$.

Finally, we consider the case when G_j contains exactly one edge. There are two subcases to handle, namely, the edge could be (i,j) or (i,k). Suppose the edge is (i,k). Then, it is easy to see that P_j will set $c^{(j)}_k = 1$ when D and P_k are honest. Next we claim that with overwhelming probability $c^{(j)}_i$ will be set to 0. Indeed, in order to force $c^{(j)}_i$ to be 1, a corrupt P_i must send some $M^{(i,j)}_{j,\ell}$ such that $\ell \notin S_{j,i}$, and $M^{(i,j)}_{j,\ell}$ is a MAC on $s^{(i,j)}_j \neq s^{(k,j)}_j = s_j$ that is consistent with key $K^{(i)}_{j,\ell}$. Note that $\{K^{(i)}_{j,\ell}\}_{\ell \notin S_{j,i}}$ is unknown to P_i since it is generated by honest D, and sent to P_j via point-to-point channels, which P_j then sends it to P_k again via point-to-point channels. Since corrupt P_i can forge this MAC only with probability negligible in σ, the claim follows. Finally, we consider the subcase when the edge is (i,j). Clearly, in this case, the value $c^{(j)}_j$ is set to 1. (Note $c^{(j)}_j$'s value depends only on the MACs and keys sent by honest D to honest P_k.) Next we claim that with overwhelming probability $c^{(j)}_i$ will be set to 0. As before, in order to force $c^{(j)}_i$ to be 1, a corrupt P_i must send some $M^{(i,j)}_{k,\ell}$ such that $\ell \notin S_{k,i}$, and $M^{(i,j)}_{k,\ell}$ is a MAC on $s^{(i,j)}_k \neq s^{(j)}_k = s_k$ that is consistent with the key $K^{(i)}_{k,\ell}$. Note that $\{K^{(i)}_{k,\ell}\}_{\ell \notin S_{k,i}}$ is unknown to P_i since it is generated by honest D, and sent to P_k via point-to-point channels, which P_k then sends it to P_j again via point-to-point channels. Since corrupt P_i can forge this MAC only with probability negligible in σ, the claim follows. Thus we conclude that the simulation is statistically indistinguishable from the real execution.

E More Details on 2-Round 4-Party Statistically Secure Protocol for Linear Functions

For simplicity, and without loss of generality, we assume that all parties wish to evaluate the same function f on their joint inputs. Let $f = \bigoplus_{k \in [4]} \alpha_k s_k$ where s_k is party P_k's input, and $\alpha_k \in \{0,1\}$ are the (publicly known) coefficients.

E.1 Protocol Description

Description of subroutines. Our protocol makes use of a variety of subroutines that we describe below. The first subroutine $\mathsf{LinRecInput}_k^{(i,j)}$ reconstructs P_k's input from the CNF shares $v_k^{(i)}$, $v_k^{(j)}$ possessed respectively by parties P_i and P_j. As with all our subroutines, this will be executed inside a PSM protocol for which P_i and P_j act as clients. Since we are dealing with malicious parties, the subroutine may get as inputs inconsistent CNF shares. In this case, the subroutine simply outputs $\perp$. If this is not the case, then the subroutine reconstructs P_k's input by xor-ing the individual shares. Thus, this subroutines serves two purposes: (1) first it reconstructs the parties' inputs on which the function can then be evaluated, and (2) it reveals whether parties received/supplied consistent shares (i.e., depending on whether the output is $\perp$ or not).

<u>SUBROUTINE $\mathsf{LinRecInput}_k^{(i,j)}(v_k^{(i)}, v_k^{(j)})$</u>

- Inputs: $v_k^{(i)} = \{(k, t, s_{k,t}^{(i)})\}_{t\in T_{i,k}}$ and $v_k^{(j)} = \{(k, t, s_{k,t}^{(j)})\}_{t\in T_{j,k}}$.
- Let $m \in [4] \setminus \{i, j, k\}$. If $s_{k,m}^{(i)} \neq s_{k,m}^{(j)}$, output $\perp$ and terminate.
- Output $s_{k,i}^{(j)} \oplus \bigoplus_{t\in T_{k,i}} s_{k,t}^{(i)}$.

The next subroutine $\mathsf{LinRecView}_k^{(i,j)}$ is useful in applying the view reconstruction trick (that we previously employed in Section 3). Since parties verifiably secret share their inputs in the first round, each party possesses 1-private 3-party CNF shares of every other parties' inputs. Given two such CNF shares it is not only possible to reconstruct the secret (this is what was done by the subroutine $\mathsf{LinRecInput}$ described above) but also allows reconstructing the other CNF share. This is exactly what $\mathsf{LinRecView}_k^{(i,j)}$ does. It obtains CNF shares of each parties' input from P_i and P_j. As we are dealing with malicious parties, the subroutine first performs a sanity check as to whether P_i and P_j supply consistent shares. If not, then it simply aborts. Else, it reconstructs the shares that ought to be held by party P_k. As we will see later, this subroutine will be immensely helpful in allowing parties to construct the inconsistency graphs. Recall that each party P_k could potentially receive PSM outputs from three PSM executions. As we will see, computing the final output from these outputs is a nontrivial task. We will need the inconsistency graphs (generated using outputs of the PSM protocols) to help us in computing the final output.

<u>SUBROUTINE $\mathsf{LinRecView}_k^{(i,j)}(v_i, v_j)$</u>

- Inputs: $v_i = \{(m, t, s_{m,t}^{(i)})\}_{m\in[4],t\in T_{i,m}}$ and $v_j = \{(m, t, s_{m,t}^{(j)})\}_{m\in[4],t\in T_{j,m}}$.
- For all $m \in [4]$ and $t \in T_{m,k}$, do the following:
  - If $t \in T_{m,i} \cap T_{m,j}$ and $s_{m,t}^{(i)} = s_{m,t}^{(j)}$, then set $\mathsf{sh}_{m,t} = s_{m,t}^{(i)}$.
  - Else if $t \in T_{m,i} \cap T_{m,j}$ and $s_{m,t}^{(i)} \neq s_{m,t}^{(j)}$, then output $\perp$ and terminate.
  - Else if $t \in T_{m,j}$, then set $\mathsf{sh}_{m,t} = s_{m,t}^{(i)}$.
  - Else if $t \in T_{m,i}$, then set $\mathsf{sh}_{m,t} = s_{m,t}^{(j)}$.

- Output $v_k^{(i,j)} = \{(m, t, \mathsf{sh}_{m,t})\}_{m\in[4], t\in T_{m,k}}$.

The final subroutine $\mathsf{SimExtract}$ that we use in our protocol is also the subroutine that the simulator uses to extract the corrupt party's input. Let $m \in [4]$. In the simulation, the simulator will set m to be the index of the corrupt party. In the real protocol, a party will invoke this procedure when it is clear that P_m is the corrupt party. The procedure $\mathsf{SimExtract}_m$ takes as input all values that were received from P_m by the remaining parties in round 1 of the protocol. Then, it constructs the inconsistency graph G' adding edges between vertices if the CNF shares held by them are not consistent. If the graph contains all three edges, then the effective input used in this case is 0. We call this the *identifiable triple-edge* case since it is clear that P_m is corrupt. Next, if the graph contains two edges or no edges (i.e., an even number of edges), then we are now assured that there exists a pair of parties that hold consistent CNF shares of P_m's input. In this case, the effective input extracted equals the secret reconstructed from these consistent CNF shares. Since it is possible that P_m may behave honestly, we call this case the *resolvable even-edge* case. As was the case in VSS, if G' contains a single-edge then the procedure performs a vote computation step using the MAC values and the corresponding keys. This is to find out which of the two parties is supported by P_m. If there is a unique party that is supported by P_m, then the inconsistency in CNF shares is resolved by using the CNF share possessed by this party. We call this the *resolvable single-edge* case. On the other hand if there is no unique party supported by P_m, then it is clear that P_m is corrupt. We call this the *identifiable single-edge* case. In this case, the effective input used for P_m equals the xor of all unique shares (including the inconsistent CNF shares) possessed by all remaining parties.

We remark that the extraction procedure is identical to the VSS extraction procedure except in the identifiable single-edge case. While in VSS, it was possible to simply output 0 in the identifiable single-edge case, things are a bit more trickier in the linear function evaluation setting. Specifically we were not able to replace the corrupt party's input by 0 and then evaluate the function while simultaneously preserving privacy of honest inputs. Fortunately though, if we use the effective input extracted as described above, then we can force all parties to compute their output that is consistent with the extracted corrupt input.

Subroutine $\mathsf{SimExtract}_m(\{w_{p,m}\}_{p\in T_m})$

- Inputs: For all $p \in T_m$, value $w_{p,m} = (\{s_{m,t}\}_{t\in T_{m,p}}, \{M^{(p)}_{m,t,\ell}\}_{t\in T_{m,p},\ell\in[\sigma]}, \{K^{(t)}_{m,p,\ell}\}_{t\in T_{m,p},\ell\in[\sigma]})$.
- Construct inconsistency graph G' such that it contains an edge between vertices $i, j \in T_m$ iff $\exists k \in [4] \setminus \{m, i, j\}$ such that $s^{(i)}_{m,k} \neq s^{(j)}_{m,k}$.
- If G' contains exactly one edge, say (i, j): Let $k \in [4] \setminus \{m, i, j\}$. For each $t \in \{i, j\}$, initialize $c_t = 0$, then pick random $S_t \subset [\sigma]$ of size $\sigma/2$, and set $c_t = 1$ if (1) $\forall \ell \in S_t$ it holds that $M^{(t)}_{m,k,\ell}$ is a MAC on $s^{(t)}_{m,k}$ that is consistent with key $K^{(t)}_{m,k,\ell}$, and (2) $\exists \ell \in [\sigma] \setminus S_t$ such that $M^{(t)}_{m,k,\ell}$ is a MAC on $s^{(t)}_{m,k}$ that is consistent with key $K^{(t)}_{m,k,\ell}$.

  - (Identifiable single-edge) If $c_i = c_j$ then output $s'_m = s^{(k)}_{m,i} \oplus s^{(k)}_{m,j} \oplus s^{(i)}_{m,k} \oplus s^{(j)}_{m,k}$.
  - (Resolvable single-edge) Else output $s'_m = s^{(k)}_{m,i} \oplus s^{(k)}_{m,j} \oplus s^{(t)}_{m,k}$ where $t \in \{i, j\}$ such that $c_t = 1$.
- (Resolvable even-edge) Else if $\exists (i, j) \notin G'$, output $s'_m = s^{(j)}_{m,i} \oplus s^{(j)}_{m,k} \oplus s^{(i)}_{m,j}$ where $k \in [4] \setminus \{m, i, j\}$.
- (Identifiable triple-edge) Else (i.e., G' contains all three edges), output $s'_m = 0$.

We are now ready to describe the complete protocol for 2-round 4-party statistically secure linear function evaluation. (See Appendix E.2 for a detailed overview and intuition behind the design of the protocol.)

Protocol. Let T_i denote the set $[4] \setminus \{i\}$, and let $T_{i,j}$ denote the set $[4] \setminus \{i, j\}$. Let $f = \bigoplus_{k \in [4]} \alpha_k s_k$ where s_k is party P_k's input, and $\alpha_k \in \{0, 1\}$ are the (publicly known) coefficients.

Round 1. For each $m \in [4]$, party P_m does the following:

- P_m holding private input s_m performs a 1-private 3-party CNF sharing of s_m among the remaining 3 parties. More precisely, it chooses random $\{s_{m,j}\}_{j \neq m}$ such that $\bigoplus_{j \neq m} s_{m,j} = s_m$, and sends CNF share $\{s^{(j)}_{m,t} = s_{m,t}\}_{t \in T_{m,j}}$ to party P_j for each $j \neq m$.
- P_m creates σ information-theoretic MACs for each value $s_{m,j}$ as $\{M^{(i)}_{m,j,\ell}, K^{(i)}_{m,j,\ell}\}_{i \in T_{m,j}, \ell \in [\sigma]}$ and sends $\{M^{(i)}_{m,j,\ell}\}_{\ell \in [\sigma]}$ to P_i for each $i \in T_{m,j}$, and $\{K^{(i)}_{m,j,\ell}\}_{i \in T_{m,j}, \ell \in [\sigma]}$ to P_j.
- P_m exchanges randomness with each P_j for a 2-client PSM protocol described below.

Round 2.

- Each pair of parties (P_i, P_j) runs the following PSM protocol $\pi^k_{i,j}$ that delivers output to P_k:
  - Inputs: $w_p = \{(\{s^{(p)}_{m,t}\}_{t \in T_{m,p}}, \{M^{(p)}_{m,t,\ell}\}_{t \in T_{m,p}, \ell \in [\sigma]}, \{K^{(t)}_{m,p,\ell}\}_{t \in T_{m,p}, \ell \in [\sigma]})\}_{m \in [4]}$ from P_p for $p = i, j$.
  - For all $\ell \in \{i, j\}$: (1) For all $m \in [4]$, set $v^{(\ell)}_m = \{(m, t, s^{(\ell)}_{m,t})\}_{t \in T_{i,m}}$. (2) Set $v_\ell = \cup_{m \in [4]} v^{(\ell)}_m$.
  - For all $m \in [4]$, compute $s'_m = \mathsf{LinRecInput}^{(i,j)}_m(v^{(i)}_m, v^{(j)}_m)$.
  - If $s'_m = \perp$ for $m \in \{i, j, k\}$ then output $\perp$.
  - Else if $s'_m = \perp$ for $m \notin \{i, j, k\}$ then output (w_i, w_j).
  - Else, output $(z^{(k)}_{i,j}, v^{(i,j)}_k)$, where $z^{(k)}_{i,j} = f(s'_1, \ldots, s'_4)$ and $v^{(i,j)}_k = \mathsf{LinRecView}^{(i,j)}_k(v_i, v_j)$.
- For $m \in [4]$ and for each $j \in T_m$, party P_j does the following for each $i \in T_{m,j}$:
  - P_j chooses a random subset $S_{m,j,i} \subset [\sigma]$ of size $\sigma/2$, and sends $(S_{m,j,i}, \{K^{(i)}_{m,j,\ell}\}_{\ell \in S_{m,j,i}})$ to P_i, and $(S_{m,j,i}, \{K^{(i)}_{m,j,\ell}\}_{\ell \in [\sigma]})$ to P_k for $k \in [4] \setminus \{i, j, m\}$.
  - P_j sends $\{M^{(j)}_{m,i,\ell}\}_{\ell \in [\sigma]}$ to P_i over point-to-point channels.

Output Computation. For $k \in [4]$, party P_k reconstructs its output as follows.

1. For $m \in T_k$: Initialize the inconsistency graph $G_k^{(m)}$ to the empty graph. Let $i, j \in T_{m,k}$ with $i \neq j$.
   - Add edge (i,j) to $G_k^{(m)}$ iff $\pi_{i,j}^k$ outputs (w_i, w_j) (i.e., with $s_{m,k}^{(i)} \neq s_{m,k}^{(j)}$).
   - Add edge (j,k) to $G_k^{(m)}$ iff $\pi_{i,j}^k$ outputs either (1) (w_i, w_j) with $s_{m,i}^{(j)} \neq s_{m,i}^{(k)}$, or (2) $(z_{i,j}^k, v_k^{(i,j)})$ with $s_{m,i}^{(j)} \neq s_{m,i}^{(k)}$, where $(m, i, s_{m,i}^{(j)}) \in v_k^{(i,j)}$.
   - Add edge (i,k) to $G_k^{(m)}$ iff $\pi_{i,j}^k$ outputs either (1) (w_i, w_j) with $s_{m,j}^{(i)} \neq s_{m,j}^{(k)}$, or (2) $(z_{i,j}^k, v_k^{(i,j)})$ with $s_{m,j}^{(i)} \neq s_{m,j}^{(k)}$, where $(m, j, s_{m,j}^{(i)}) \in v_k^{(i,j)}$.
2. If $\exists m \in T_k$ such that $G_k^{(m)}$ contains 3 edges, say $(i,j), (j,k), (i,k)$, then
   - Assert that output of $\pi_{i,j}^k$ equals (w_i, w_j).
   - Parse w_i, w_j to obtain for all $p \in T_m$ and $\ell \in \{i,j\}$ the set $v_p^{(\ell)} = \{(p, t, s_{p,t}^{(\ell)})\}_{t \in T_{\ell,p}}$.
   - Set $s'_m = 0$. For each $p \in T_m$, set $s'_p = \mathsf{LinRecInput}_p^{(i,j)}(v_p^{(i)}, v_p^{(j)})$.
   - Output $z_k = f(s'_1, \ldots, s'_4)$ and terminate.
3. For each $m \in T_k$ such that $G_k^{(m)}$ contains exactly one edge, say (i,j) with $m' \in [4] \setminus \{i,j,m\}$ (note that it is possible that $k \in \{i,j\}$), then:
   - Initialize $c_{m,i}^{(k)} = c_{m,j}^{(k)} = 0$.
   - If $k \in \{i,j\}$, then set $c_{m,k}^{(k)} = 1$ if $\forall \ell \in S_{m,m',k}$ it holds that $M_{m,m',\ell}^{(k)}$ is a MAC on $s_{m,m'}^{(k)}$ consistent with key $K_{m,m',\ell}^{(k)}$.
   - For $m'' \in \{i,j\} \setminus \{k\}$, set $c_{m,m''}^{(k)} = 1$ if (1) $\forall \ell \in S_{m,m',m''}$ it holds that $M_{m,m',\ell}^{(m'',k)}$ is a MAC on $s_{m,m'}^{(m'')}$ consistent with key $K_{m,m',\ell}^{(m'')}$, and (2) $\exists \ell \in [\sigma] \setminus S_{m,m',m''}$ such that $M_{m,m',\ell}^{(m'',k)}$ is a MAC on $s_{m,m'}^{(m'')}$ consistent with key $K_{m,m',\ell}^{(m'')}$.
4. If $\exists m \in T_k$ such that $G_k^{(m)}$ contains exactly one edge, say (i,j), and if $c_{m,i}^{(k)} = c_{m,j}^{(k)}$, then
   - If $k \in \{i,j\}$: Let $m' \in [4] \setminus \{i,j,m\}$ and $m'' \in \{i,j\} \setminus \{k\}$.
     - Assert that output of $\pi_{m',m''}^k$ equals $(z_{m',m''}^k, v_k^{(m',m'')})$. If not output fail_1 and terminate.
     - Output $z'_k = z_{m',m''}^k \oplus \alpha_m s_{m,m'}^{(k)}$ and terminate.
   - Else if $k \notin \{i,j\}$:
     - Assert that output of $\pi_{i,j}^k$ equals (w_i, w_j). If not output fail_1 and terminate.
     - Parse w_i, w_j to obtain for all $p \in T_m$ and $\ell \in \{i,j\}$ the set $v_p^{(\ell)} = \{(p, t, s_{p,t}^{(\ell)})\}_{t \in T_{\ell,p}}$.
     - For all $p \in T_m$, set $s'_p = \mathsf{LinRecInput}_p^{(i,j)}(v_p^{(i)}, v_p^{(j)})$.
     - Compute $s'_m = s_{m,i}^{(k)} \oplus s_{m,j}^{(k)} \oplus s_{m,k}^{(i)} \oplus s_{m,k}^{(j)}$.
     - Output $z'_k = \bigoplus_{p \in [4]} \alpha_p s'_p$ and terminate.
5. Construct the *accusation graph* A_k as follows: Initialize A_k as the 4-vertex empty graph.
   - For each $m \in T_k$, if there are two edges (i,m'), (j,m') in $G_k^{(m)}$, then add edge (m,m') to A_k.

- For each $m \in T_k$, if there is exactly one edge (i, j) in $G_k^{(m)}$, then add edge (i, m) to A_k if $c_{m,i}^{(k)} = 0$, else add edge (j, m).

6. If A_k contains no edges, then:
   - Assert that there exists $i, j \in T_k$ such that $\pi_{i,j}^k$ outputs $(z_{i,j}^k, v_k^{(i,j)})$ for some $z_{i,j}^k$, $v_k^{(i,j)}$.
   - Let i, j be from the previous step. Output $z_k' = z_{i,j}^k$ and terminate.
7. Else if A_k contains exactly one edge (m, i) for some $m, i \in T_k$, then let $j \in [4] \setminus \{m, i, k\}$.
   - If $\exists m' \in \{m, i\}$ s.t. $\pi_{m',j}^k$ outputs $(w_{m'}, w_j)$, then
     - Parse w_j to obtain for all $p \in [4]$ the set $v_p^{(j)} = \{(p, t, s_{p,t}^{(j)})\}_{t \in T_{j,p}}$.
     - For each $p \in [4]$, set $s_p' = \mathsf{LinRecInput}_p^{(j,k)}(v_p^{(j)}, v_p^{(k)})$.
     - Output $z_k' = f(s_1', \ldots, s_4')$, and terminate.
   - Else assert that there exists $m', m'' \in \{m, i\}$ with $m' \neq m''$ such that the output of $\pi_{m',j}^k$ equals $(z_{m',j}^k, v_k^{(m',j)})$ and further that $v_k^{(m',j)}$ satisfies $v_k^{(m',j)} \setminus \{(m'', j, \mathsf{sh}_{m'',j})\} = v_k \setminus \{(m'', j, s_{m'',j}^{(k)})\}$.
     - Output $z_{m',j}^k \oplus \alpha_{m''}(\mathsf{sh}_{m'',j} \oplus s_{m'',j}^{(k)})$.
8. Else if A_k contains the edge (m, k) for some $m \in T_k$, or A_k contains two edges (m, i) and (m, j) for some $i, j, m \in T_k$, then:
   - If $\pi_{i,j}^k$ outputs (w_i, w_j), then:
     - Parse w_j to obtain for all $p \in [4]$ the set $v_p^{(j)} = \{(p, t, s_{p,t}^{(t)})\}_{t \in T_{j,p}}$.
     - For each $p \in T_m$, set $s_p' = \mathsf{LinRecInput}_p^{(j,k)}(v_p^{(j)}, v_p^{(k)})$.
     - Set $w_{k,m} = (\{s_{m,t}^{(k)}\}_{t \in T_{m,k}}, \{M_{m,t,\ell}^{(k)}\}_{t \in T_{m,k}, \ell \in [\sigma]}, \{K_{m,k,\ell}^{(t)}\}_{t \in T_{m,k}, \ell \in [\sigma]})$.
     - For $p \in \{i, j\}$, parse w_p to obtain $w_{p,m} = (\{s_{m,t}^{(p)}\}_{t \in T_{m,p}}, \{M_{m,t,\ell}^{(p)}\}_{t \in T_{m,p}, \ell \in [\sigma]}, \{K_{m,p,\ell}^{(t)}\}_{t \in T_{m,p}, \ell \in [\sigma]})$.
     - Compute $s_m' = \mathsf{SimExtract}_m(\{w_{p,m}\}_{p \in T_m})$.
     - Output $z_k' = f(s_1', \ldots, s_4')$, and terminate.
   - Else assert that the output of $\pi_{i,j}^k$ is $(z_{i,j}^k, v_k^{(i,j)})$ for some $z_{i,j}^k, v_k^{(i,j)}$.
     - If both (i, k) and (j, k) are contained in $G_k^{(m)}$, then output $z_{i,j}^k$.
     - Else if $\exists m', m'' \in \{i, j\}$ with $m' \neq m''$ such that $(m', k) \in G_k^{(m)}$, then
       - If $c_{m,m'}^{(k)} = 1$, output $z_{i,j}^k$.
       - Else output $z_{i,j}^k \oplus \alpha_m(s_{m,m''}^{(m')} \oplus s_{m,m''}^{(k)})$.
   - Else output $z_{i,j}^k$.

E.2 Detailed Overview and Intuition

In the first step of the protocol, each party essentially runs Step 1 of the VSS protocol (of Section 4) as the dealer, and secret shares its input. In addition, each party exchanges randomness for PSM protocols to be executed in round 2. That is:

Round 1.

- For each $m \in [4]$, party P_m does the following:

- P_m holding private input s_m performs a 1-private 3-party CNF sharing of s_m among the remaining 3 parties. More precisely, it chooses random $\{s_{m,j}\}_{j\neq m}$ such that $\bigoplus_{j\neq m} s_{m,j} = s_m$, and sends CNF share $\{s^{(j)}_{m,t} = s_{m,t}\}_{t\in T_{m,j}}$ to party P_j for each $j \neq m$.
- P_m creates σ information-theoretic MACs for each value $s_{m,j}$ as $\{M^{(i)}_{m,j,\ell}, K^{(i)}_{m,j,\ell}\}_{i\in T_{m,j},\ell\in[\sigma]}$ and sends $\{M^{(i)}_{m,j,\ell}\}_{\ell\in[\sigma]}$ to P_i for each $i \in T_{m,j}$, and $\{K^{(i)}_{m,j,\ell}\}_{i\in T_{m,j},\ell\in[\sigma]}$ to P_j.
- P_m exchanges randomness with each P_j for a 2-client PSM protocol described below.

<u>*Simulation extraction.*</u> Next we describe the simulation extraction based on the first round messages of the adversary corrupting party P_q. We do this first since this extraction procedure will later serve as the guiding light in the design of the rest of the protocol (i.e., the second round, and the output computation phase). In particular, the simulation extraction procedure will dictate what outputs the honest parties receive in the ideal execution. Obviously, we will need to design the remaining steps of our protocol in a way that allows honest parties to compute the exact same output (i.e., one that's consistent with the input extracted by the simulator) in the real execution. The simulation extraction procedure will help us in identifying what values are needed to compute the final output in the real execution. and will guide the design of the PSM executions and the 2nd round messages such that the honest parties indeed obtain these values (while preserving privacy against the adversary). We defer further discussion, and focus now on the simulation extraction procedure itself. Focusing on the corrupt party, say P_q, (ignoring the randomness exchanged for executing the PSM protocols) we have that P_q sends the following in round 1:

1. P_q sends $\{s^{(j)}_{q,t} = s_{q,t}\}_{t\in T_{q,j}}$ to party P_j for each $j \neq q$.
2. $\forall j \in T_q$, party P_q sends $\{M^{(i)}_{q,j,\ell}\}_{\ell\in[\sigma]}$ to P_i for each $i \in T_{q,j}$, and $\{K^{(i)}_{q,j,\ell}\}_{i\in T_{q,j},\ell\in[\sigma]}$ to P_j.

We describe the simulation extraction procedure below. In the simulation, the simulator will invoke $\mathsf{SimExtract}_q$, i.e., with $m = q$.

<u>SUBROUTINE $\mathsf{SimExtract}_m(\{w_{p,m}\}_{p\in T_m})$</u>

– Inputs: For all $p \in T_m$, value $w_{p,m} = (\{s_{m,t}\}_{t\in T_{m,p}}, \{M^{(p)}_{m,t,\ell}\}_{t\in T_{m,p},\ell\in[\sigma]}, \{K^{(t)}_{m,p,\ell}\}_{t\in T_{m,p},\ell\in[\sigma]})$.
– Construct inconsistency graph G' such that it contains an edge between vertices $i, j \in T_m$ iff $\exists k \in [4] \setminus \{m, i, j\}$ such that $s^{(i)}_{m,k} \neq s^{(j)}_{m,k}$.
– If G' contains exactly one edge, say (i, j): Let $k \in [4] \setminus \{m, i, j\}$. For each $t \in \{i, j\}$, initialize $c_t = 0$, then pick random $S_t \subset [\sigma]$ of size $\sigma/2$, and set $c_t = 1$ if (1) $\forall \ell \in S_t$ it holds that $M^{(t)}_{m,k,\ell}$ is a MAC on $s^{(t)}_{m,k}$ that is consistent with key $K^{(t)}_{m,k,\ell}$, and (2) $\exists \ell \in [\sigma] \setminus S_t$ such that $M^{(t)}_{m,k,\ell}$ is a MAC on $s^{(t)}_{m,k}$ that is consistent with key $K^{(t)}_{m,k,\ell}$.

  - (Identifiable single-edge) If $c_i = c_j$ then output $s'_m = s^{(k)}_{m,i} \oplus s^{(k)}_{m,j} \oplus s^{(i)}_{m,k} \oplus s^{(j)}_{m,k}$.
  - (Resolvable single-edge) Else output $s'_m = s^{(k)}_{m,i} \oplus s^{(k)}_{m,j} \oplus s^{(t)}_{m,k}$ where $t \in \{i, j\}$ such that $c_t = 1$.
- (Resolvable even-edge) Else if $\exists (i, j) \notin G'$, output $s'_m = s^{(j)}_{m,i} \oplus s^{(j)}_{m,k} \oplus s^{(i)}_{m,j}$ where $k \in [4] \setminus \{m, i, j\}$.
- (Identifiable triple-edge) Else (i.e., G' contains all three edges), output $s'_m = 0$.

In the simulation, the simulator will set $m = q$, i.e, to the index of the corrupt party. The procedure $\mathsf{SimExtract}_m$ takes as input all values that were received from P_m by the remaining parties in round 1 of the protocol. Then, it constructs the inconsistency graph G' adding edges between vertices if the CNF shares held by them are not consistent. If the graph contains all three edges, then the effective input used in this case is 0. We call this the *identifiable triple-edge* case since in this case it will be clear to the remaining honest parties that P_m is corrupt. Next, if the graph contains two edges or no edges (i.e., an even number of edges), then we are now assured that there exists a pair of parties that hold consistent CNF shares of P_m's input. In this case, the effective input extracted equals the secret reconstructed from these consistent CNF shares. Since it is possible that other honest parties may not be convinced that P_m is corrupt, we call this case the *resolvable even-edge* case. As was the case in VSS, if G' contains a single-edge then the procedure performs a vote computation step using the MAC values and the corresponding keys. This is to find out which of the two parties is supported by P_m. If there is a unique party that is supported by P_m, then the inconsistency in CNF shares is resolved by using the CNF share possessed by this party. We call this the *resolvable single-edge* case. On the other hand if there is no unique party supported by P_m, then it will become clear to all parties that P_m is corrupt. We call this the *identifiable single-edge* case. In this case, the effective input used for P_m equals the xor of all unique shares (including the inconsistent CNF shares) possessed by all remaining parties.

We remark that the extraction procedure is identical to the VSS extraction procedure except in the identifiable single-edge case. While in VSS, it was possible to simply output 0 in the identifiable single-edge case, things are a bit more trickier in the linear function evaluation setting. Specifically we were not able to replace the corrupt party's input by 0 and then evaluate the function while simultaneously preserving privacy of honest inputs. As we will see later, fortunately enough, if we use the effective input extracted as described above, then we can force all parties to compute their output that is consistent with the extracted corrupt input.

Designing the PSM executions. As described earlier, we will use PSM protocols to help the parties evaluate the function f. More precisely, party P_k acts as the PSM referee and obtains the PSM outputs from PSM execution $\pi^k_{i,j}$ for each distinct $i, j \in T_k$, where parties P_i and P_j act as the PSM clients. (I.e., each P_k obtains outputs from three PSM executions.) To see why pairwise PSMs suffice, observe that the input of each party is 1-private CNF shared between the remaining parties, and thus, two parties may come together to reconstruct the secret. Of course, this secret reconstruction cannot be done in the clear since this violates privacy. However this can be done inside the PSM protocol. Specifically, the PSM protocol will use the following subroutine:

<u>SUBROUTINE $\mathsf{LinRecInput}_k^{(i,j)}(v_k^{(i)}, v_k^{(j)})$</u>

- Inputs: $v_k^{(i)} = \{(k, t, s_{k,t}^{(i)})\}_{t \in T_{i,k}}$ and $v_k^{(j)} = \{(k, t, s_{k,t}^{(j)})\}_{t \in T_{j,k}}$.
- Let $m \in [4] \setminus \{i, j, k\}$. If $s_{k,m}^{(i)} \neq s_{k,m}^{(j)}$, output $\perp$ and terminate.
- Output $s_{k,i}^{(j)} \oplus \bigoplus_{t \in T_{k,i}} s_{k,t}^{(i)}$.

It is easy to see that the above procedure reconstructs a non-$\perp$ value only if P_i and P_j supply consistent CNF shares of P_k's input. In this case, the reconstruction is carried out in the standard way. Thus, if shares are distributed consistently by the malicious party, then the above subroutine helps to reconstruct each parties' inputs after which the function can be evaluated inside the PSM execution.

In addition to the above, we will also use the PSM protocols to help the parties construct the inconsistency graphs (analogous to the ones used in the VSS protocol). Recall that each party P_k could potentially receive PSM outputs from three PSM executions. As we will see, computing the final output from these outputs is a non-trivial task. We will need the inconsistency graphs (generated using outputs of the PSM protocols) to help us in computing the final output. Towards helping us construct these inconsistency graphs, we will have the following subroutine executed inside each PSM execution.

<u>SUBROUTINE $\mathsf{LinRecView}_k^{(i,j)}(v_i, v_j)$</u>

- Inputs: $v_i = \{(m, t, s_{m,t}^{(i)})\}_{m \in [4], t \in T_{i,m}}$ and $v_j = \{(m, t, s_{m,t}^{(j)})\}_{m \in [4], t \in T_{j,m}}$.
- For all $m \in [4]$ and $t \in T_{m,k}$, do the following:
  - If $t \in T_{m,i} \cap T_{m,j}$ and $s_{m,t}^{(i)} = s_{m,t}^{(j)}$, then set $\mathsf{sh}_{m,t} = s_{m,t}^{(i)}$.
  - Else if $t \in T_{m,i} \cap T_{m,j}$ and $s_{m,t}^{(i)} \neq s_{m,t}^{(j)}$, then output $\perp$ and terminate.
  - Else if $t \in T_{m,j}$, then set $\mathsf{sh}_{m,t} = s_{m,t}^{(i)}$.
  - Else if $t \in T_{m,i}$, then set $\mathsf{sh}_{m,t} = s_{m,t}^{(j)}$.
- Output $v_k^{(i,j)} = \{(m, t, \mathsf{sh}_{m,t})\}_{m \in [4], t \in T_{m,k}}$.

Note how the efficient extendability of the CNF secret sharing scheme enables us to reconstruct the shares ought to be held by P_k (i.e., using the CNF shares held jointly by P_i and P_j). Note however that P_i and P_j may not hold (or supply) consistent shares, in which case $\mathsf{LinRecView}_k^{(i,j)}$ delivers $\perp$ as output to P_k. (Actually, we will use this subroutine only when the shares jointly held by both P_i and P_j are consistent.) When all parties are honest, it should be clear that $\mathsf{LinRecView}_k^{(i,j)}$ only provides P_k what it already knows. (Note in particular that the MACs associated with the CNF shares are not handled by the subroutine.) Further observe that a malicious P_k cannot in any way force to learn additional information from $\mathsf{LinRecView}_k^{(i,j)}$.

We are now ready to describe the PSM subprotocol that makes use of the two subroutines described above, and then how the output of the PSM executions helps P_k construct the inconsistency graphs. The PSM subprotocol $\pi_{i,j}^k$ takes inputs from P_i and P_j and delivers outputs to P_k in the following way:

- Inputs: $w_p = \{(\{s_{m,t}^{(p)}\}_{t\in T_{m,p}}, \{M_{m,t,\ell}^{(p)}\}_{t\in T_{m,p},\ell\in[\sigma]}, \{K_{m,p,\ell}^{(t)}\}_{t\in T_{m,p},\ell\in[\sigma]})\}_{m\in[4]}$ from P_p for $p = i, j$.
- For all $\ell \in \{i,j\}$: (1) For all $m \in [4]$, set $v_m^{(\ell)} = \{(m,t,s_{m,t}^{(\ell)})\}_{t\in T_{i,m}}$. (2) Set $v_\ell = \cup_{m\in[4]} v_m^{(\ell)}$.
- For all $m \in [4]$, compute $s'_m = \mathsf{LinRecInput}_m^{(i,j)}(v_m^{(i)}, v_m^{(j)})$.
- If $s'_m = \perp$ for $m \in \{i,j,k\}$ then output $\perp$.
- Else if $s'_m = \perp$ for $m \notin \{i,j,k\}$ then output (w_i, w_j).
- Else, output $(z_{i,j}^{(k)}, v_k^{(i,j)})$, where $z_{i,j}^{(k)} = f(s'_1, \ldots, s'_4)$ and $v_k^{(i,j)} = \mathsf{LinRecView}_k^{(i,j)}(v_i, v_j)$.

<u>*Privacy guarantees.*</u> Obviously we will need *robust* PSM protocols to implement $\pi_{i,j}^k$ in order to guarantee security against a malicious client. Next, we claim that when the PSM referee, party P_k in this case, is malicious, then $\pi_{i,j}^k$ does not violate privacy. Specifically, in this case, it will hold that $s'_m \neq \perp$ for all $m \in T_k$ (and therefore the output will never be (w_i, w_j)). If $s'_k = \perp$, then output of $\pi_{i,j}^k$ is $\perp$, and privacy clearly holds. On the other hand, if the output equals $(z_{i,j}^k, v_k^{(i,j)})$, then we use the fact that the output $v_k^{(i,j)}$ from $\mathsf{LinRecView}_k^{(i,j)}$ does not violate privacy. (Recall that $\mathsf{LinRecView}_k^{(i,j)}$ provides P_k only what it already knows from the first round.) Now it remains to be shown that value $z_{i,j}^k$ does not provide to P_k any information besides the evaluation of the linear function. The argument is slightly trickier since P_k could potentially set things up (e.g., by providing inconsistent shares) in a way such that the three PSM executions provide it evaluations of the function f on different choices of P_k's input. It is here that we rely on the fact that f is a linear function, and make use of the property that given an evaluation z of f on a set of points $\{s_m\}_{m\in T_k}$ and s_k, it is possible to obtain an evaluation z' of f on a set of points $\{s_m\}_{m\in T_k}$ and s'_k for any choice of s'_k. Given the above property of linear functions, it follows that privacy is preserved against the malicious party.

<u>*Constructing the inconsistency graphs.*</u> Next, we explain how (an honest) party P_k can construct the inconsistency graphs. Specifically, party P_k uses the output of $\pi_{i,j}^k$ to construct the inconsistency graph $G_k^{(m)}$ for $m \in [4] \setminus \{i,j,k\}$ as follows:

- Add edge (i,j) to $G_k^{(m)}$ iff $\pi_{i,j}^k$ outputs (w_i, w_j) (i.e., with $s_{m,k}^{(i)} \neq s_{m,k}^{(j)}$).
- Add edge (j,k) to $G_k^{(m)}$ iff $\pi_{i,j}^k$ outputs either (1) (w_i, w_j) with $s_{m,i}^{(j)} \neq s_{m,i}^{(k)}$, or (2) $(z_{i,j}^k, v_k^{(i,j)})$ with $s_{m,i}^{(j)} \neq s_{m,i}^{(k)}$, where $(m, i, s_{m,i}^{(j)}) \in v_k^{(i,j)}$.

– Add edge (i,k) to $G_k^{(m)}$ iff $\pi_{i,j}^k$ outputs either (1) (w_i, w_j) with $s_{m,j}^{(i)} \neq s_{m,j}^{(k)}$, or (2) $(z_{i,j}^k, v_k^{(i,j)})$ with $s_{m,j}^{(i)} \neq s_{m,j}^{(k)}$, where $(m, j, s_{m,j}^{(i)}) \in v_k^{(i,j)}$.

As in the VSS protocol, these inconsistency graphs contain edges between two parties only if they hold inconsistent shares (and in particular, does not depend on the validity of MACs). It should be clear that two honest parties will not have an edge between them in $G_k^{(m)}$ as long as P_m is honest. We also point out that two different honest parties, say P_i and P_j may hold different inconsistency graphs $G_i^{(m)} \neq G_j^{(m)}$.

<u>*Towards handling the single-edge case.*</u> Looking ahead, we will need the parties to make use of the MAC values to handle the single-edge case (and in particular to decide whether it is identifiable or resolvable). Note that while in the VSS protocol, each party P_j would send all its shares $\{s_{m,t}^{(j)}\}_{t \in T_{m,j}}$ and all the MACs $\{M_{m,t,\ell}^{(j)}\}_{t \in T_{m,i}}$ to each P_i with $i \in T_{m,j}$, it is not a good idea to do so when parties wish to evaluate a linear function. Doing this, i.e., leaking the CNF shares would essentially leak each parties input. Further note that if a party that possesses the keys also learns the (information-theoretic) MACs, then this once again would violate privacy. (Note that this is acceptable in the case of VSS as the second round was the "reconstruction phase" in which all parties are expected to learn the dealer's secret.) This motivates a careful design of the step where parties exchange MACs and keys while (1) preserving privacy of the shares and (2) allowing parties to handle the single-edge case. We formally describe this step which is run in parallel with the robust PSM executions in round 2.

– For $m \in [4]$ and for each $j \in T_m$, party P_j does the following for each $i \in T_{m,j}$:
  - P_j chooses a random subset $S_{m,j,i} \subset [\sigma]$ of size $\sigma/2$, and sends $(S_{m,j,i}, \{K_{m,j,\ell}^{(i)}\}_{\ell \in S_{m,j,i}})$ to P_i, and $(S_{m,j,i}, \{K_{m,j,\ell}^{(i)}\}_{\ell \in [\sigma]})$ to P_k for $k \in [4] \setminus \{i, j, m\}$.
  - P_j sends $\{M_{m,i,\ell}^{(j)}\}_{\ell \in [\sigma]}$ to P_i over point-to-point channels.

This completes the description of round 2. For the sake of clarity, we present the full description of protocol steps executed in round 2.

<u>*Round 2.*</u>

– Each pair of parties (P_i, P_j) runs the following PSM protocol $\pi_{i,j}^k$ that delivers output to P_k:
  - Inputs: $w_p = \{(\{s_{m,t}^{(p)}\}_{t \in T_{m,p}}, \{M_{m,t,\ell}^{(p)}\}_{t \in T_{m,p}, \ell \in [\sigma]}, \{K_{m,p,\ell}^{(t)}\}_{t \in T_{m,p}, \ell \in [\sigma]})\}_{m \in [4]}$ from P_p for $p = i, j$.
  - Set $v_i = \cup_{m \in [4]} v_m^{(i)} = \cup_{m \in [4]} \{(m, t, s_{m,t}^{(i)})\}_{t \in T_{i,m}}$ and $v_j = \cup_{m \in [4]} v_m^{(j)} = \cup_{m \in [4]} \{(m, t, s_{m,t}^{(j)})\}_{t \in T_{j,m}}$.
  - For all $m \in [4]$, compute $s'_m = \mathsf{LinRecInput}_m^{(i,j)}(v_m^{(i)}, v_m^{(j)})$.
  - If $s'_m = \perp$ for $m \in \{i, j, k\}$ then output $\perp$.

- Else if $s'_m = \perp$ for $m \notin \{i, j, k\}$ then output (w_i, w_j).
- Else, output $(z_{i,j}^{(k)}, v_k^{(i,j)})$, where $z_{i,j}^{(k)} = f(s'_1, \ldots, s'_4)$ and $v_k^{(i,j)} = \mathsf{LinRecView}_k^{(i,j)}(v_i, v_j)$.

– For $m \in [4]$ and for each $j \in T_m$, party P_j does the following for each $i \in T_{m,j}$:
- P_j chooses a random subset $S_{m,j,i} \subset [\sigma]$ of size $\sigma/2$, and sends $(S_{m,j,i}, \{K_{m,j,\ell}^{(i)}\}_{\ell \in S_{m,j,i}})$ to P_i, and $(S_{m,j,i}, \{K_{m,j,\ell}^{(i)}\}_{\ell \in [\sigma]})$ to P_k for $k \in [4] \setminus \{i, j, m\}$.
- P_j sends $\{M_{m,i,\ell}^{(j)}\}_{\ell \in [\sigma]}$ to P_i over point-to-point channels.

Now it remains to show how to design the output computation phase that ensures that each party P_k computes the correct output, i.e., one that is consistent with the simulation. The analysis below handles the identifiable cases and the resolvable cases separately. As pointed out earlier, the identifiable cases are relatively easier to handle, and so we focus on that first.

Handling the identifiable cases. Recall that in the identifiable cases, the inconsistency graph $G_k^{(m)}$ for corrupt P_m contains either (1) all three edges, or (2) a single edge with vote parity 0. (Note we show how to compute the votes below.) We handle the identifiable triple-edge case as follows:

– If $\exists m \in T_k$ such that $G_k^{(m)}$ contains 3 edges, say $(i, j), (j, k), (i, k)$, then
  - Assert that output of $\pi_{i,j}^k$ equals (w_i, w_j).
  - Parse w_i, w_j to obtain for all $p \in T_m$ and $\ell \in \{i, j\}$ the set $v_p^{(\ell)} = \{(p, t, s_{p,t}^{(\ell)})\}_{t \in T_{\ell,p}}$.
  - Set $s'_m = 0$. For each $p \in T_m$, set $s'_p = \mathsf{LinRecInput}_p^{(i,j)}(v_p^{(i)}, v_p^{(j)})$.
  - Output $z_k = f(s'_1, \ldots, s'_4)$ and terminate.

  To see why the above works, recall that for an honest P_m, the graph $G_k^{(m)}$ will never contain all three edges. This is because there is always a pair of honest parties P_i and P_j (other than P_m) that hold consistent shares distributed by P_m, and hence do not have an edge between them in $G_k^{(m)}$. Therefore if there exists $m \in T_k$ such that $G_k^{(m)}$ contains all three edges, then P_k can be assured that P_m is corrupt. Further, since $(i, j) \in G_k^{(m)}$ it must hold that $\pi_{i,j}^k$ output (w_i, w_j) (i.e., the protocol will never terminate with output fail_3). Next, note that for each $t \in T_m$, party P_t is honest, and that honest P_i and P_j hold consistent shares of s_t. Thus $\mathsf{LinRecInput}_t^{(i,j)}$ will reconstruct the correct output, i.e., $s'_t = s_t$. Note that we have substituted the corrupt party's input with 0, exactly as done in the simulation. Thus the output of P_k will be z_k in both the real and ideal executions.

Next we focus on handling the identifiable single-edge case, i.e., the inconsistency graph $G_k^{(m)}$ contains a single edge (i, j) with vote parity 0. To set things up, we first need to ensure that party P_k can indeed compute the votes (and the vote parity). Thus, we describe how P_k computes the votes first, and then describe how P_k computes the final output depending on the votes (rather vote parity) and the output of the PSM executions.

Computing the votes. Ideally we want the vote calculation procedure to be exactly as in the VSS protocol (and this will also help ensure that the distribution of the votes is statistically close to the simulation). The main difficulty in this setting is that P_k may not always explicitly know the secret, say s_m (e.g., in cases where P_m is honest), or even the MACs (since knowing the keys as well as MACs corresponding to secret s_m will reveal s_m).

To design the vote computing mechanism, we will need to consider two cases. First, suppose $k \in \{i, j\}$. In this case, our protocol's second round (non-PSM) messages are designed in a way to enable computation of $c^{(k)}_{m,k}$. More specifically, P_k receives $(S_{m,m',k}, \{K^{(k)}_{m,m',\ell}\}_{\ell \in S_{m,m',k}})$ from $P_{m'}$ where $m' \in [4] \setminus \{i, j, m\}$. These values along with $M^{(k)}_{m,m',\ell}$ that P_k already received from P_m are sufficient to let P_k to compute $c^{(k)}_{m,k}$. Next, suppose $m'' \in \{i, j\} \setminus \{k\}$. Once again, computing $c^{(k)}_{m,m''}$ is made possible by receiving the relevant values from $P_{m'}$ (and also MAC values received from $P_{m''}$ over point-to-point channels), but this works only if $m' \neq k$ (which is exactly the case when $k \in \{i, j\}$). When $m' = k$, i.e., $k \notin \{i, j\}$, party P_k would need to know the MAC values $M^{(m'')}_{m,m',\ell}$ in order to compute $c^{(k)}_{m,m''}$. Fortunately in this case, we will be able to leverage the output of the PSM executions. Specifically, when $(i, j) \in G^{(m)}_k$, in this case the output of the PSM execution $\pi^k_{i,j}$ must be (w_i, w_j) (with $s^{(i)}_{m,k} \neq s^{(j)}_{m,k}$). Thus, now P_k can parse (w_i, w_j) to learn the relevant MAC values, denoted $M^{(m'',k)}_{m,m',\ell}$ (which equals $M^{(m'')}_{m,m',\ell}$ when $P_{m''}$ is honest). We now describe the vote computation step concretely.

- For each $m \in T_k$ such that $G^{(m)}_k$ contains exactly one edge, say (i, j) with $m' \in [4] \setminus \{i, j, m\}$ (note that it is possible that $k \in \{i, j\}$), then P_k does the following:
  - Party P_k initializes $c^{(k)}_{m,i} = c^{(k)}_{m,j} = 0$.
  - If $k \in \{i, j\}$, then party P_k sets $c^{(k)}_{m,k} = 1$ if $\forall \ell \in S_{m,m',k}$ it holds that $M^{(k)}_{m,m',\ell}$ is a MAC on $s^{(k)}_{m,m'}$ consistent with key $K^{(k)}_{m,m',\ell}$.
  - For $m'' \in \{i, j\} \setminus \{k\}$, party P_k sets $c^{(k)}_{m,m''} = 1$ if (1) $\forall \ell \in S_{m,m',m''}$ it holds that $M^{(m'',k)}_{m,m',\ell}$ is a MAC on $s^{(m'')}_{m,m'}$ consistent with key $K^{(m'')}_{m,m',\ell}$, and (2) $\exists \ell \in [\sigma] \setminus S_{m,m',m''}$ such that $M^{(m'',k)}_{m,m',\ell}$ is a MAC on $s^{(m'')}_{m,m'}$ consistent with key $K^{(m'')}_{m,m',\ell}$.

Computing the output. Given that the votes are computed as above, we now describe how P_k computes its output in the identifiable single-edge case (i.e., vote parity equals 0). We begin by first noting that party P_m must be corrupt in this case. Thus, all the remaining parties will hold the same inconsistency graph as P_k, i.e., all parties hold the inconsistency graph $G^{(m)}_k$ (that contains the single edge (i, j)). Furthermore, the vote computation step above is designed in a way such that for all $t_1, t_2, p \in T_m$, it will hold that $c_{m,p}^{(t_1)} = c_{m,p}^{(t_2)}$ with all but negligible probability. Now the steps required to compute the final output will take advantage of the above facts to ensure that all parties compute the same final output. As it turns out, designing these steps is somewhat challenging since honest parties may not all have access to the corrupt party's (effective) input.

To better understand this challenge, let us look at the views of all three parties other than corrupt P_m. Let $m' \in [4] \setminus \{i, j, m\}$, and thus we denote the three parties as P_i, P_j, and $P_{m'}$. Note that each of $P_i, P_j, P_{m'}$ must be in a position to compute the final output using the output of a single PSM execution which is guaranteed to deliver output. (The other two PSM executions involve corrupt P_m as client, and may simply abort.) Somewhat counterintuitively this simplifies the challenge in the sense that the parties can simply discard the outcome of PSM executions which involve corrupt P_m as client. We begin the analysis by looking at the easy case where party $P_{m'}$ tries to reconstruct the output using the output of the PSM execution where P_i and P_j are the clients. Since $(i, j) \in G_{m'}^{(m)}$ it must hold that the output of $\pi_{i,j}^{m'}$ equals (w_i, w_j). Obviously, (honest) $P_{m'}$ can now reconstruct each honest parties' input in the following way: for all $p \in T_m$, set $s'_p = \mathsf{LinRecInput}_p^{(i,j)}(v_p^{(i)}, v_p^{(j)})$. These reconstructed inputs are guaranteed to be correct since both P_i and P_j are honest parties, and thus hold consistent shares of every honest parties' input. Now $P_{m'}$ is in a position to compute the correct output (irrespective of what decision proecedure we finally use) since it possesses $w_i, w_j, w_{m'}$ (which will be used to extract the correct effective input for corrupt P_m), and also the values $\{s'_p\}_{p \in T_m}$. It remains to be shown how exactly is P_m's effective input extracted, and for this we will need to first look at what parties P_i, P_j can reconstruct.

Consider party P_i. We will use the fact that $G_i^{(m)}$ contains exactly one edge (i.e., (i, j)), and therefore, the output of $\pi_{m',j}^{i}$ will not equal $(w_{m'}, w_j)$. In other words, we are guaranteed that the output of $\pi_{m',j}^{i}$ equals $(z_{m',j}^{i}, v_i^{(m',j)})$. Our first observation is that the value $z_{m',j}^{i}$ is computed using the honest inputs (because both $P_{m'}$ and P_j are honest and thus provide correct shares of all honest inputs). Next, note that the effective input of corrupt P_m used to compute $z_{m',j}^{i}$ would be $\tilde{s}_m^{(i)} = s_{m,i}^{(m')} \oplus s_{m,j}^{(m')} \oplus s_{m,m'}^{(j)}$ (i.e., using CNF shares possessed by $P_{m'}$ and P_j). That is, the value $z_{m',j}^{i}$ is computed using values $\{s'_p\}_{p \in T_m}$ and $\tilde{s}_m^{(i)}$. By an analogous argument, we have that P_j receives from $\pi_{m',i}^{j}$ output $(z_{m',i}^{j}, v_j^{(m',i)})$ where the value $z_{m',i}^{j}$ is computed using values $\{s'_p\}_{p \in T_m}$ and $\tilde{s}_m^{(j)} = s_{m,i}^{(m')} \oplus s_{m,j}^{(m')} \oplus s_{m,m'}^{(i)}$. Note that $\tilde{s}_m^{(i)} \neq \tilde{s}_m^{(j)}$ since $s_{m,m'}^{(j)} \neq s_{m,m'}^{(i)}$ (and this is precisely why (i, j) is an edge in the inconsistency graph). Thus, it remains to be shown how P_i and P_j can compute the same final output in this case.

A first attempt would be to let P_i and P_j (and also $P_{m'}$) to try and substitute corrupt P_m's input by 0 in the function evaluation. However, in order to do this, party P_i (resp. P_j) would first need to cancel out the shares corresponding to P_m's input from the value $z_{m',j}^{i}$ (resp. $z_{m',i}^{j}$). Unfortunately, in order to protect privacy against corrupt PSM referees, the protocol is designed in a way such that

P_i would not learn $\tilde{s}_m^{(i)}$ (in particular the missing share $s_{m,i}^{(m')}$) from $v_i^{(m',j)}$, and therefore cannot cancel out $\alpha_m \tilde{s}_m^{(i)}$ from $z_{m',j}^i$. This poses a major hindrance to our plan of computing the final output by simply replacing corrupt P_m's input by 0.

We overcome the obstacle by using the following trick. Instead of attempting to substitute corrupt P_m's input by 0, we let P_i to locally "correct" the output $z_{m',j}^i$ by XORing it with $\alpha_m s_{m,m'}^{(i)}$. (Note party P_i obtains CNF share $s_{m,m'}^{(i)}$ from corrupt P_m in round 1 of the protocol.) Likewise P_j locally "corrects" $z_{m',i}^j$ by XORing it with $\alpha_m s_{m,m'}^{(j)}$. This has the effect of ensuring that both P_i and P_j agree on the same final output (*without* knowing what effective input of corrupt P_m they use). To see why let $\Gamma = (\bigoplus_{p \in T_m} \alpha_p s'_p)$. Then, observe that $z_{m',j}^i = \Gamma \oplus \alpha_m \tilde{s}_m^{(i)}$ and that $z_{m',i}^j = \Gamma \oplus \alpha_m \tilde{s}_m^{(j)}$. Thus, to prove agreement, it suffices to show that $\tilde{s}_m^{(i)} \oplus s_{m,m'}^{(i)} = \tilde{s}_m^{(j)} \oplus s_{m,m'}^{(j)}$. This in fact, follows immediately upon inspection. (Recall $\tilde{s}_m^{(i)} = s_{m,i}^{(m')} \oplus s_{m,j}^{(m')} \oplus s_{m,m'}^{(j)}$ while $\tilde{s}_m^{(j)} = s_{m,i}^{(m')} \oplus s_{m,j}^{(m')} \oplus s_{m,m'}^{(i)}$.) To conclude in this case, the effective corrupt input used equals $s_{m,i}^{(m')} \oplus s_{m,j}^{(m')} \oplus s_{m,m'}^{(i)} \oplus s_{m,m'}^{(j)}$. Thus, we have the following steps:

- If $\exists m \in T_k$ such that $G_k^{(m)}$ contains exactly one edge, say (i,j), and if $c_{m,i}^{(k)} = c_{m,j}^{(k)}$, then
  - If $k \in \{i,j\}$: Let $m' \in [4] \setminus \{i,j,m\}$ and $m'' \in \{i,j\} \setminus \{k\}$.
    - Assert that output of $\pi_{m',m''}^k$ equals $(z_{m',m''}^k, v_k^{(m',m'')})$. If not output fail_1 and terminate.
    - Output $z'_k = z_{m',m''}^k \oplus \alpha_m s_{m,m'}^{(k)}$ and terminate.
  - Else if $k \notin \{i,j\}$:
    - Assert that output of $\pi_{i,j}^k$ equals (w_i, w_j). If not output fail_1 and terminate.
    - Parse w_i, w_j to obtain for all $p \in T_m$ and $\ell \in \{i,j\}$ the set $v_p^{(\ell)} = \{(p,t,s_{p,t}^{(\ell)})\}_{t \in T_{\ell,p}}$.
    - For all $p \in T_m$, set $s'_p = \mathsf{LinRecInput}_p^{(i,j)}(v_p^{(i)}, v_p^{(j)})$.
    - Compute $s'_m = s_{m,i}^{(k)} \oplus s_{m,j}^{(k)} \oplus s_{m,k}^{(i)} \oplus s_{m,k}^{(j)}$.
    - Output $z'_k = \bigoplus_{p \in [4]} \alpha_p s'_p$ and terminate.

Handling the resolvable cases. Now we are in the remaining cases where it is clear that in the ideal execution, the output of the honest parties, say z', is computed using the adversary input, say s' that is extracted from consistent secret shares possessed by a pair of honest parties. Obviously now it remains to be shown that honest parties will output z' even in the real execution. Towards this, we let each party P_k construct an *accusation graph* A_k using the inconsistency graphs $\{G_k^{(m)}\}_{m \in T_k}$. We will ensure the following property for honest P_k: Graph A_k contains an edge between two vertices i and j iff one of P_i, P_j is dishonest. The accusation graph is constructed as follows:

1. For each $m \in T_k$, if there are two edges $(i, m'), (j, m')$ in $G_k^{(m)}$, then add edge (m, m') to A_k.
2. For each $m \in T_k$, if there is exactly one edge (i, j) in $G_k^{(m)}$, then add edge (i, m) to A_k if $c_{m,i}^{(k)} = 0$, else add edge (j, m).

In the first case, clearly one of $P_m, P_{m'}$ is corrupt. This is because if P_i (resp. P_j) was corrupt, then (j, m') (resp. (i, m')) cannot be an edge in $G_k^{(m)}$. In the second case, suppose edge (i, m) was added to A_k, then we argue that one of P_i, P_m is corrupt. (The case when (j, m) is added is handled similarly.) This is because if $P_{m'}$ with $m' \in [4] \setminus \{i, j, m\}$ was corrupt then (i, j) cannot be an edge in $G_k^{(m)}$. On the other hand if P_j was corrupt (and therefore $P_i, P_m, P_{m'}$ are honest), then $c_{m,i}^{(k)} = 1$ always holds. Thus, we conclude that graph A_k constructed as above contains an edge between two vertices i, j iff one of P_i, P_j is corrupt.

Now we describe how the accusation graph is used to ensure that the final output of P_k is computed as z'. We analyze this case-by-case depending on the structure of A_k.

1. A_k contains no edges. In this case, P_k outputs $z_{i,j}^k$ for any $i, j \in T_k$.
   Our main claim for handling this case is that for every $i, j \in T_k$, the output of $\pi_{i,j}^k$ equals $(z_{i,j}^k = z', v_k^{(i,j)} = v_k)$. (Recall z' is the output in the ideal execution.) Our starting observation is that if there are no edges in A_k, then for every $m \in T_k$ there must be no edges in $G_k^{(m)}$. (This is true since (1) the case when $G_k^{(m)}$ has three edges was already handled, and (2) the case when $G_k^{(m)}$ has two edges always results in A_k having at least one edge, and (3) the case when $G_k^{(m)}$ has a single edge is either already handled (i.e., in the identifiable case) or results in A_k having an edge (i.e., in the resolvable case).) This immediately allows us to show that for every $i, j \in T_k$, the output of $\pi_{i,j}^k$ must be of the form $(*, v_k^{(i,j)} = v_k)$, since if $v_k^{(i,j)} \neq v_k$, then there would be an edge (either (i, k) or (j, k)) in $G_k^{(m)}$ for $m \in [4] \setminus \{i, j, k\}$. Now conditioned on $v_k^{(i,j)} = v_k$, it immediately follows that honest P_k's shares for each parties' input (including the adversary's) are consistent with those held by the honest parties among P_i, P_j. Thus, (1) the honest inputs used to compute $z_{i,j}^k$ equal the honest parties' actual inputs, and (2) the adversary input used to compute the output $z_{i,j}^k$ equals the value s' that can be extracted from consistent shares of P_k and the honest parties among P_i, P_j. Thus, we conclude that $z_{i,j}^k = z'$ must hold.
2. A_k contains exactly one edge (m, i) for some $m, i \in T_k$.
   Let $j \in [4] \setminus \{i, m, k\}$. Since A_k contains (m, i), it must hold that one of P_m, P_i is dishonest, and so we have that P_j is honest. The challenge now is to choose how to compute the final output depending on the outputs of $\pi_{m',j}^k$ for $m' \in \{m, i\}$. (Note that P_k can safely ignore the output of $\pi_{i,m}^k$.)
   First observe that for all $m' \in \{m, i\}$, the graph $G_k^{(m')}$ does not contain the edge (j, k). This is because if $(j, k) \in G_k^{(m')}$, then either (m', j) or (m', k)

would be an edge in A_k. (Note that either (1) $G_k^{(m')}$ contains two edges, or (2) $G_k^{(m')}$ contains a resolvable single-edge. In either case, it holds that either (m', j) or (m', k) would be an edge in A_k.) Since we have that A_k contains exactly one edge (m, i), the observation follows. Since (j, k) is not an edge in $G_k^{(m')}$ for $m' \in \{m, i\}$, we have that P_j and P_k hold consistent shares of the corrupt party. Thus:

– If for some $m' \in \{m, i\}$, protocol $\pi^k_{m',j}$ outputs $(w_{m'}, w_j)$, then P_k can compute the correct output simply by setting $s'_t = \mathsf{LinRecInput}_t^{(j,k)}(v_t^{(j)}, v_t^{(k)})$ for all $t \in [4]$ and output $z' = f(s'_1, \ldots, s'_4)$.

The next case to handle is when for all $m' \in \{m, i\}$ the output of $\pi^k_{m',j}$ equals $(z^k_{m',j}, v_k^{(m',j)})$. How do we decide which output to accept? How do we break the symmetry? In fact, it is not even clear if the symmetry can be broken. Luckily as it turns out we don't have to break the symmetry, and loosely speaking, we turn the situation on its head and design a procedure such that it that extracts the same final output (consistent with the ideal execution) from either $\pi^k_{i,j}$ or $\pi^k_{m,j}$ (i.e., even when $(z^k_{i,j}, v_k^{(i,j)}) \neq (z^{(k)}_{m,j}, v_k^{(m,j)})$). We first assert the following:

– Assert that there exists $m', m'' \in \{m, i\}$ with $m' \neq m''$ such that the output of $\pi^k_{m',j}$ equals $(z^k_{m',j}, v_k^{(m',j)})$ and further that $v_k^{(m',j)}$ satisfies $v_k^{(m',j)} \setminus \{(m'', j, \mathsf{sh}_{m'',j})\} = v_k \setminus \{(m'', j, s^{(k)}_{m'',j})\}$.
  - Let $P_{m'}$ with $m' \in \{m, i\}$ be honest. We will prove that the assertion holds for $P_{m'}$. Next, note (1) $\pi^k_{m',j}$ will not output $\perp$ since $P_{m'}, P_j, P_k$ are all honest, and (2) $\pi^k_{m',j}$ did not output $(w_{m'}, w_j)$ since if it did then the protocol would have already terminated after being handled in the case above. Thus, it must be the case that $\pi^k_{m',j}$ output $(z^k_{m',j}, v_k^{(m',j)})$.

    Now let us look at what happens inside subroutine $\mathsf{LinRecView}_k^{(i,j)}$. Since $P_{m'}$ and P_j are both honest, they will obviously agree on all shares $s_{p,t}$ where $p \in \{m', j, k\}$ (i.e., the honest parties. When $p = m''$ (i.e., the index corresponding to the corrupt party), note that the subroutine $\mathsf{LinRecView}_k^{(i,j)}$ does not perform any equality tests, and therefore does not return $\perp$. Therefore, let $v_k^{(m',j)} = \{(p, t, \mathsf{sh}_{p,t})\}_{p \in [4], t \in T_{p,k}}$. Clearly for all $p \in \{m', j, k\}$, we have that $(p, t, \mathsf{sh}_{p,t}) = (p, t, s^{(k)}_{p,t})$. Now $\mathsf{sh}_{m'',m'}$ equals $s^{(j)}_{m'',m'}$ (i.e., is held by P_j). If $s^{(j)}_{m'',m'} \neq s^{(k)}_{m'',m'}$, then $(j, k) \in G_k^{(m)}$ must hold – a contradiction. Thus, we have that $\mathsf{sh}_{m'',m'} = s^{(k)}_{m'',m'}$. This suffices to prove that the output of $\pi^k_{m',j}$ equals $(z^k_{m',j}, v_k^{(m',j)})$ and further that $v_k^{(m',j)}$ satisfies $v_k^{(m',j)} \setminus \{(m'', j, \mathsf{sh}_{m'',j})\} = v_k \setminus \{(m'', j, s^{(k)}_{m'',j})\}$.

For the rest of the analysis, assume wlog that $m' = i$. That is, from the above assertion we have that either the output of $\pi^k_{i,j}$ equals $(z^k_{i,j}, v_k^{(i,j)})$ with either (1) $v_k^{(i,j)} = v_k$, or (2) $v_k^{(i,j)}$ differing from v_k only in the value $s^{(k)}_{m,j}$.

– Suppose that output of $\pi^k_{i,j}$ equals $(z^k_{i,j}, v^{(i,j)}_k)$, and $v^{(i,j)}_k = v_k$: Then P_k outputs $z^k_{i,j}$.

  - Note already that $(j,k) \notin G^{(m)}_k$. Since $\pi^k_{i,j}$ did not output (w_i, w_j), it also holds that $(i,j) \notin G^{(m)}_k$. Now if $v^{(i,j)}_k = v_k$, then even $(i,k) \notin G^{(m)}_k$. That is, we have that $G^{(m)}_k$ is the empty graph. If P_m is corrupt, then the correct decision is to accept $z^k_{i,j}$ (since the effective input used for the corrupt party is indeed constructed using consistent shares possessed by the honest parties). On the other hand, if P_m is honest, then we are assured that the true input of P_m is used to compute $z^k_{i,j}$. However, it is not clear if $z^k_{i,j}$ was computed using input of (corrupt) P_i that is consistent with the simulation. To show this we use the fact that $(j,k) \notin G^{(i)}_k$. (Recall that (j,k) is neither an edge in $G^{(m)}_k$ nor in $G^{(i)}_k$.) Thus the output was computed using the input of (corrupt) P_i that is consistent with the shares held by P_j and P_k, and is therefore, consistent with the simulation as well.

– Suppose that output of $\pi^k_{i,j}$ equals $(z^k_{i,j}, v^{(i,j)}_k)$, but $v^{(i,j)}_k \neq v_k$ (i.e., $s^{(i)}_{m,j} \neq s^{(k)}_{m,j}$): If $c^{(k)}_{m,k} = 0$, output $z^k_{i,j}$, else output $z^k_{i,j} \oplus \alpha_m (s^{(i)}_{m,j} \oplus s^{(k)}_{m,j})$.

  - Recall that P_j is honest. In this case, $(i,j) \notin G^{(m)}_k$ since otherwise the output of $\pi^k_{i,j}$ would have been (w_i, w_j). Also, recall that $(j,k) \notin G^{(m)}_k$. Note that $s^{(i)}_{m,j} \neq s^{(k)}_{m,j}$, and so we conclude that $(i,k) \in G^{(m)}_k$. Therefore, we have a single-edge in $G^{(m)}_k$, and we consider the values $c^{(k)}_{m,i}$ and $c^{(k)}_{m,k}$. If $c^{(k)}_{m,i} = 1$ (and $c^{(k)}_{m,k} = 0$), then the correct decision is to accept $z^k_{i,j}$, since in this case it is clear that P_m is corrupt. Since P_i and P_j are honest, then it is immediate that $z^k_{i,j}$ equals the output computed using the adversary input as extracted by the simulator. On the other hand if $c^{(k)}_{m,k} = 1$ (and $c^{(k)}_{m,i} = 0$), then the value $z^k_{i,j}$ is computed using an adversary input that is different from the one used in the simulation. Here we take advantage of the fact that f is a linear function, and that given an evaluation of linear function at a specific point x, it is possible to obtain an evaluation at a different point x' as long as x, x' differ in exactly one coordinate and the difference is known. Specifically, for $f(s'_1, \ldots, s'_4) = \bigoplus_{t \in [4]} \alpha_t s'_t$, we now compute the final output as $z_k = z^k_{i,j} + \alpha_m (s^{(i)}_{m,j} \oplus s^{(k)}_{m,j})$. It remains to be shown that this step indeed computes the correct output.
    - ∗ We first consider the case when P_i is honest (i.e., P_m is corrupt). Note that the $z^k_{i,j} = f(s''_1, \ldots, s''_4)$ where for all $t \in T_m$, it holds that s''_t equals the true input of the honest party P_t, and $s''_m = s^{(j)}_{m,i} \oplus s^{(j)}_{m,k} \oplus s^{(i)}_{m,j}$ equals the value reconstructed using the shares $s^{(j)}_{m,i}, s^{(j)}_{m,k}$ provided by party P_j, and the share $s^{(i)}_{m,j}$ provided by party P_i. Since $c^{(k)}_{m,i} = 0$ the simulator in the ideal execution will reconstruct the adversary's input using the shares held by P_j and

P_k, i.e., $s'_m = s^{(j)}_{m,i} \oplus s^{(j)}_{m,k} \oplus s^{(k)}_{m,j}$. I.e., $z' = (\bigoplus_{t \in T_m} \alpha_t s''_t) \oplus \alpha_m s'_m$ while $z^k_{i,j} = (\bigoplus_{t \in T_m} \alpha_t s''_t) \oplus \alpha_m s''_m$. Thus, we let P_k apply the correction $\beta^{(i,k)}_{m,j} = \alpha_m(s^{(i)}_{m,j} \oplus s^{(k)}_{m,j})$ to $z^k_{i,j}$ to obtain z'. Note that it is possible to apply the correction since P_k already possesses $s^{(k)}_{m,j}$ and it obtains $s^{(i)}_{m,j,}$ as this is part of $v^{(i,j)}_k$.

∗ Finally, we consider the case when P_i is corrupt (i.e., P_m is honest). In this case, we need to show that the procedure outlined above still produces the correct output z'. More precisely, we need to show that applying the following steps produces the correct output: (1) if $c^{(k)}_{m,k} = 0$, then output $z^k_{i,j}$, (2) else output $z^k_{i,j} \oplus \alpha_m(s^{(i)}_{m,j} \oplus s^{(k)}_{m,j})$. Actually, since P_m and P_j are honest, the event that $c^{(k)}_{m,k} = 0$ never occurs. This allows us to focus only on the second case. Now, we have $c^{(k)}_{m,i} = 0$, and yet P_i might somehow ensure that $\pi^k_{i,j}$ outputs $z^k_{i,j} = z'$. (Note ironically this is problematic since our procedure applies the correction $\beta^{(i,k)}_{m,j}$ to $z^k_{i,j} = z'$ thereby resulting in incorrect output $z' + \beta^{(i,k)}_{m,j}$.) The key observation is that such an attack cannot be carried out by corrupt P_i. This follows from our main assertion that the values v_k and $v^{(i,j)}_k$ can differ only in the CNF share $s_{m,j}$, i.e., $s^{(i)}_{m,j} \neq s^{(k)}_{m,j}$, and from the observation that $z^k_{i,j}$ is computed using values supplied by P_i that appear in $v^{(i,j)}_k$. Fortunately, in this case, the correction $\beta^{(i,k)}_{m,j}$ applied to $z^k_{i,j}$ will result in the correct output z'.

3. A_k contains the edge (m, k) for some $m \in T_k$, or A_k contains two edges (m, i) and (m, j) for $i, j, m \in T_k$.
   In both these cases, it is clear to P_k that P_m is corrupt (i.e., P_i and P_j are honest), and that the final output has to be extracted from the output of $\pi^k_{i,j}$ (i.e., the other PSM executions can be ignored). Next, it is straightforward to handle the case when $\pi^k_{i,j}$ outputs (w_i, w_j) since:
   - If $\pi^k_{i,j}$ outputs (w_i, w_j), then P_k outputs the output of the simulation extractor on w_i, w_j, w_k.

   It is straightforward to see that P_k produces the output exactly as in the ideal execution. The harder case is when $\pi^k_{i,j}$ outputs $(z^k_{i,j}, v^{(i,j)}_k)$. Clearly $(i, j) \notin G^{(m)}_k$ since otherwise $\pi^k_{i,j}$ would have output (w_i, w_j). Now it appears that we can simply accept $z^k_{i,j}$. However, this is not correct. We additionally need to check if (i, k) or (j, k) is contained in $G^{(m)}_k$.
   - If both (i, k) and (j, k) are contained in $G^{(m)}_k$, then P_k outputs $z^k_{i,j}$.
   - Else if only $(i, k) \in G^{(m)}_k$ (wlog), then (1) output $z^k_{i,j}$ if $c^{(k)}_{m,i} = 1$, (2) else output $z^k_{i,j} \oplus \beta^{(i,k)}_{m,j}$.

   To see why the above works, note that when both (i, k) and (j, k) are contained in $G^{(m)}_k$, the simulation extractor would use the view of the honest parties P_i and P_j (who are also consistent between themselves since otherwise $\pi^k_{i,j}$ wouldn't output $(z^k_{i,j}, v^{(i,j)}_k)$) to generate the final output. Since this output would equal $z^k_{i,j}$, we can let P_k simply accept $z^k_{i,j}$ as the final output. On the other hand, when only one of the two edges, say (i,k) exists in $G^{(m)}_k$, then the simulation extractor would see which of the two parties P_i, P_k did P_m "support", i.e., depending on which of $c^{(k)}_{m,i}$, $c^{(k)}_{m,k}$, equals 1. Following this, P_k decides whether to accept $z^k_{i,j}$ (i.e., if $c^{(k)}_{m,i}$ equals 1), or apply the correction to $z^k_{i,j}$ (i.e., if $c^{(k)}_{m,k}$ equals 1). As before, applying the correction has the effect of canceling out the wrong value of share $s_{m,j}$ (i.e., $s^{(i)}_{m,j}$) and re-adding the 'right' value (i.e., $s^{(k)}_{m,j}$), and therefore ensuring that the effective input used is consistent with the value extracted in the ideal process simulation.

Summarizing, we have the following resolution procedure for each party P_k:

1. Construct the *accusation graph* A_k as follows: Initialize A_k as the 4-vertex empty graph.
   - For each $m \in T_k$, if there are two edges (i, m'), (j, m') in $G^{(m)}_k$, then add edge (m, m') to A_k.
   - For each $m \in T_k$, if there is exactly one edge (i, j) in $G^{(m)}_k$, then add edge (i, m) to A_k if $c^{(k)}_{m,i} = 0$, else add edge (j, m).
2. If A_k contains no edges, then:
   - Assert that there exists $i, j \in T_k$ such that $\pi^k_{i,j}$ outputs $(z^k_{i,j}, v^{(i,j)}_k)$ for some $z^k_{i,j}$, $v^{(i,j)}_k$.
   - Let i, j be from the previous step. Output $z'_k = z^k_{i,j}$ and terminate.
3. Else if A_k contains exactly one edge (m, i) for some $m, i \in T_k$, then let $j \in [4] \setminus \{m, i, k\}$.
   (a) If $\exists m' \in \{m, i\}$ s.t. $\pi^k_{m',j}$ outputs $(w_{m'}, w_j)$, then
      - Parse w_j to obtain for all $p \in [4]$ the set $v^{(j)}_p = \{(p, t, s^{(j)}_{p,t})\}_{t \in T_{j,p}}$.
      - For each $p \in [4]$, set $s'_p = \mathsf{LinRecInput}^{(j,k)}_p(v^{(j)}_p, v^{(k)}_p)$.
      - Output $z'_k = f(s'_1, \ldots, s'_4)$, and terminate.

   (b) Else assert that there exists $m', m'' \in \{m, i\}$ with $m' \neq m''$ such that the output of $\pi^k_{m',j}$ equals $(z^k_{m',j}, v^{(m',j)}_k)$ and further that $v^{(m',j)}_k$ satisfies $v^{(m',j)}_k \setminus \{(m'', j, \mathsf{sh}_{m'',j})\} = v_k \setminus \{(m'', j, s^{(k)}_{m'',j})\}$.
      - Output $z^k_{m',j} \oplus \alpha_{m''}(\mathsf{sh}_{m'',j} \oplus s^{(k)}_{m'',j})$.
4. Else if A_k contains the edge (m, k) for some $m \in T_k$, or A_k contains two edges (m, i) and (m, j) for some $i, j, m \in T_k$, then:
   - If $\pi^k_{i,j}$ outputs (w_i, w_j), then:
     - Parse w_j to obtain for all $p \in [4]$ the set $v^{(j)}_p = \{(p, t, s^{(t)}_{p,t})\}_{t \in T_{j,p}}$.
     - For each $p \in T_m$, set $s'_p = \mathsf{LinRecInput}^{(j,k)}_p(v^{(j)}_p, v^{(k)}_p)$.
     - Compute $s'_m = \mathsf{SimExtract}^{(i,j,k)}_m(w_i, w_j, w_k)$.

- Output $z'_k = f(s'_1, \ldots, s'_4)$, and terminate.

– Else assert that the output of $\pi^k_{i,j}$ is $(z^k_{i,j}, v^{(i,j)}_k)$ for some $z^k_{i,j}, v^{(i,j)}_k$.
  - If both (i,k) and (j,k) are contained in $G^{(m)}_k$, then output $z^k_{i,j}$.
  - Else if $\exists m', m'' \in \{i,j\}$ with $m' \neq m''$ such that $(m',k) \in G^{(m)}_k$, then
    * If $c^{(k)}_{m,m'} = 1$, output $z^k_{i,j}$.
    * Else output $z^k_{i,j} \oplus \alpha_m (s^{(m')}_{m,m''} \oplus s^{(k)}_{m,m''})$.

We are now ready to describe the complete protocol for 2-round 4-party statistically secure linear function evaluation.

Protocol. Let T_i denote the set $[4] \setminus \{i\}$, and let $T_{i,j}$ denote the set $[4] \setminus \{i,j\}$.

Round 1. For each $m \in [4]$, party P_m does the following:

– P_m holding private input s_m performs a 1-private 3-party CNF sharing of s_m among the remaining 3 parties. More precisely, it chooses random $\{s_{m,j}\}_{j \neq m}$ such that $\bigoplus_{j \neq m} s_{m,j} = s_m$, and sends CNF share $\{s^{(j)}_{m,t} = s_{m,t}\}_{t \in T_{m,j}}$ to party P_j for each $j \neq m$.
– P_m creates σ information-theoretic MACs for each value $s_{m,j}$ as $\{M^{(i)}_{m,j,\ell}, K^{(i)}_{m,j,\ell}\}_{i \in T_{m,j}, \ell \in [\sigma]}$ and sends $\{M^{(i)}_{m,j,\ell}\}_{\ell \in [\sigma]}$ to P_i for each $i \in T_{m,j}$, and $\{K^{(i)}_{m,j,\ell}\}_{i \in T_{m,j}, \ell \in [\sigma]}$ to P_j.
– P_m exchanges randomness with each P_j for a 2-client PSM protocol described below.

Round 2.

– Each pair of parties (P_i, P_j) runs the following PSM protocol $\pi^k_{i,j}$ that delivers output to P_k:
  – Inputs: $w_p = \{(\{s^{(p)}_{m,t}\}_{t \in T_{m,p}}, \{M^{(p)}_{m,t,\ell}\}_{t \in T_{m,p}, \ell \in [\sigma]}, \{K^{(t)}_{m,p,\ell}\}_{t \in T_{m,p}, \ell \in [\sigma]})\}_{m \in [4]}$ from P_p for $p = i, j$.
  – For all $\ell \in \{i,j\}$: (1) For all $m \in [4]$, set $v^{(\ell)}_m = \{(m, t, s^{(\ell)}_{m,t})\}_{t \in T_{i,m}}$. (2) Set $v_\ell = \cup_{m \in [4]} v^{(\ell)}_m$.
  – For all $m \in [4]$, compute $s'_m = \mathsf{LinRecInput}^{(i,j)}_m (v^{(i)}_m, v^{(j)}_m)$.
  – If $s'_m = \perp$ for $m \in \{i,j,k\}$ then output $\perp$.
  – Else if $s'_m = \perp$ for $m \notin \{i,j,k\}$ then output (w_i, w_j).
  – Else, output $(z^{(k)}_{i,j}, v^{(i,j)}_k)$, where $z^{(k)}_{i,j} = f(s'_1, \ldots, s'_4)$ and $v^{(i,j)}_k = \mathsf{LinRecView}^{(i,j)}_k (v_i, v_j)$.

– For $m \in [4]$ and for each $j \in T_m$, party P_j does the following for each $i \in T_{m,j}$:
  - P_j chooses a random subset $S_{m,j,i} \subset [\sigma]$ of size $\sigma/2$, and sends $(S_{m,j,i}, \{K^{(i)}_{m,j,\ell}\}_{\ell \in S_{m,j,i}})$ to P_i, and $(S_{m,j,i}, \{K^{(i)}_{m,j,\ell}\}_{\ell \in [\sigma]})$ to P_k for $k \in [4] \setminus \{i,j,m\}$.

- P_j sends $\{M^{(j)}_{m,i,\ell}\}_{\ell\in[\sigma]}$ to P_i over point-to-point channels.

Output Computation. For $k \in [4]$, party P_k reconstructs its output as follows.

1. For $m \in T_k$: Initialize the inconsistency graph $G^{(m)}_k$ to the empty graph. Let $i, j \in T_{m,k}$ with $i \neq j$.
   - Add edge (i, j) to $G^{(m)}_k$ iff $\pi^k_{i,j}$ outputs (w_i, w_j) (i.e., with $s^{(i)}_{m,k} \neq s^{(j)}_{m,k}$).
   - Add edge (j, k) to $G^{(m)}_k$ iff $\pi^k_{i,j}$ outputs either (1) (w_i, w_j) with $s^{(j)}_{m,i} \neq s^{(k)}_{m,i}$, or (2) $(z^k_{i,j}, v^{(i,j)}_k)$ with $s^{(j)}_{m,i} \neq s^{(k)}_{m,i}$, where $(m, i, s^{(j)}_{m,i}) \in v^{(i,j)}_k$.
   - Add edge (i, k) to $G^{(m)}_k$ iff $\pi^k_{i,j}$ outputs either (1) (w_i, w_j) with $s^{(i)}_{m,j} \neq s^{(k)}_{m,j}$, or (2) $(z^k_{i,j}, v^{(i,j)}_k)$ with $s^{(i)}_{m,j} \neq s^{(k)}_{m,j}$, where $(m, j, s^{(i)}_{m,j}) \in v^{(i,j)}_k$.
2. If $\exists m \in T_k$ such that $G^{(m)}_k$ contains 3 edges, say $(i, j), (j, k), (i, k)$, then
   - Assert that output of $\pi^k_{i,j}$ equals (w_i, w_j).
   - Parse w_i, w_j to obtain for all $p \in T_m$ and $\ell \in \{i, j\}$ the set $v^{(\ell)}_p = \{(p, t, s^{(\ell)}_{p,t})\}_{t\in T_{\ell,p}}$.
   - Set $s'_m = 0$. For each $p \in T_m$, set $s'_p = \mathsf{LinRecInput}^{(i,j)}_p(v^{(i)}_p, v^{(j)}_p)$.
   - Output $z_k = f(s'_1, \ldots, s'_4)$ and terminate.
3. For each $m \in T_k$ such that $G^{(m)}_k$ contains exactly one edge, say (i, j) with $m' \in [4] \setminus \{i, j, m\}$ (note that it is possible that $k \in \{i, j\}$), then:
   - Initialize $c^{(k)}_{m,i} = c^{(k)}_{m,j} = 0$.
   - If $k \in \{i, j\}$, then set $c^{(k)}_{m,k} = 1$ if $\forall \ell \in S_{m,m',k}$ it holds that $M^{(k)}_{m,m',\ell}$ is a MAC on $s^{(k)}_{m,m'}$ consistent with key $K^{(k)}_{m,m',\ell}$.
   - For $m'' \in \{i, j\} \setminus \{k\}$, set $c^{(k)}_{m,m''} = 1$ if (1) $\forall \ell \in S_{m,m',m''}$ it holds that $M^{(m'',k)}_{m,m',\ell}$ is a MAC on $s^{(m'')}_{m,m'}$ consistent with key $K^{(m'')}_{m,m',\ell}$, and (2) $\exists \ell \in [\sigma] \setminus S_{m,m',m''}$ such that $M^{(m'',k)}_{m,m',\ell}$ is a MAC on $s^{(m'')}_{m,m'}$ consistent with key $K^{(m'')}_{m,m',\ell}$.
4. If $\exists m \in T_k$ such that $G^{(m)}_k$ contains exactly one edge, say (i, j), and if $c^{(k)}_{m,i} = c^{(k)}_{m,j}$, then
   - If $k \in \{i, j\}$: Let $m' \in [4] \setminus \{i, j, m\}$ and $m'' \in \{i, j\} \setminus \{k\}$.
     - Assert that output of $\pi^k_{m',m''}$ equals $(z^k_{m',m''}, v^{(m',m'')}_k)$. If not output fail_1 and terminate.
     - Output $z'_k = z^k_{m',m''} \oplus \alpha_m s^{(k)}_{m,m'}$ and terminate.
   - Else if $k \notin \{i, j\}$:
     - Assert that output of $\pi^k_{i,j}$ equals (w_i, w_j). If not output fail_1 and terminate.
     - Parse w_i, w_j to obtain for all $p \in T_m$ and $\ell \in \{i, j\}$ the set $v^{(\ell)}_p = \{(p, t, s^{(\ell)}_{p,t})\}_{t\in T_{\ell,p}}$.
     - For all $p \in T_m$, set $s'_p = \mathsf{LinRecInput}^{(i,j)}_p(v^{(i)}_p, v^{(j)}_p)$.
     - Compute $s'_m = s^{(k)}_{m,i} \oplus s^{(k)}_{m,j} \oplus s^{(i)}_{m,k} \oplus s^{(j)}_{m,k}$.
     - Output $z'_k = \bigoplus_{p\in[4]} \alpha_p s'_p$ and terminate.

5. Construct the *accusation graph* A_k as follows: Initialize A_k as the 4-vertex empty graph.
   - For each $m \in T_k$, if there are two edges (i, m'), (j, m') in $G_k^{(m)}$, then add edge (m, m') to A_k.
   - For each $m \in T_k$, if there is exactly one edge (i, j) in $G_k^{(m)}$, then add edge (i, m) to A_k if $c_{m,i}^{(k)} = 0$, else add edge (j, m).
6. If A_k contains no edges, then:
   - Assert that there exists $i, j \in T_k$ such that $\pi_{i,j}^k$ outputs $(z_{i,j}^k, v_k^{(i,j)})$ for some $z_{i,j}^k$, $v_k^{(i,j)}$.
   - Let i, j be from the previous step. Output $z'_k = z_{i,j}^k$ and terminate.
7. Else if A_k contains exactly one edge (m, i) for some $m, i \in T_k$, then let $j \in [4] \setminus \{m, i, k\}$.
   (a) If $\exists m' \in \{m, i\}$ s.t. $\pi_{m',j}^k$ outputs $(w_{m'}, w_j)$, then
      - Parse w_j to obtain for all $p \in [4]$ the set $v_p^{(j)} = \{(p, t, s_{p,t}^{(j)})\}_{t \in T_{j,p}}$.
      - For each $p \in [4]$, set $s'_p = \mathsf{LinRecInput}_p^{(j,k)}(v_p^{(j)}, v_p^{(k)})$.
      - Output $z'_k = f(s'_1, \ldots, s'_4)$, and terminate.

   (b) Else assert that there exists $m', m'' \in \{m, i\}$ with $m' \neq m''$ such that the output of $\pi_{m',j}^k$ equals $(z_{m',j}^k, v_k^{(m',j)})$ and further that $v_k^{(m',j)}$ satisfies $v_k^{(m',j)} \setminus \{(m'', j, \mathsf{sh}_{m'',j})\} = v_k \setminus \{(m'', j, s_{m'',j}^{(k)})\}$.
      - Output $z_{m',j}^k \oplus \alpha_{m''}(\mathsf{sh}_{m'',j} \oplus s_{m'',j}^{(k)})$.
8. Else if A_k contains the edge (m, k) for some $m \in T_k$, or A_k contains two edges (m, i) and (m, j) for some $i, j, m \in T_k$, then:
   - If $\pi_{i,j}^k$ outputs (w_i, w_j), then:
     - Parse w_j to obtain for all $p \in [4]$ the set $v_p^{(j)} = \{(p, t, s_{p,t}^{(t)})\}_{t \in T_{j,p}}$.
     - For each $p \in T_m$, set $s'_p = \mathsf{LinRecInput}_p^{(j,k)}(v_p^{(j)}, v_p^{(k)})$.
     - Set $w_{k,m} = (\{s_{m,t}^{(k)}\}_{t \in T_{m,k}}, \{M_{m,t,\ell}^{(k)}\}_{t \in T_{m,k}, \ell \in [\sigma]}, \{K_{m,k,\ell}^{(t)}\}_{t \in T_{m,k}, \ell \in [\sigma]})$.
     - For $p \in \{i, j\}$, parse w_p to obtain $w_{p,m} = (\{s_{m,t}^{(p)}\}_{t \in T_{m,p}}, \{M_{m,t,\ell}^{(p)}\}_{t \in T_{m,p}, \ell \in [\sigma]}, \{K_{m,p,\ell}^{(t)}\}_{t \in T_{m,p}, \ell \in [\sigma]})$.
     - Compute $s'_m = \mathsf{SimExtract}_m(\{w_{p,m}\}_{p \in T_m})$.
     - Output $z'_k = f(s'_1, \ldots, s'_4)$, and terminate.
   - Else assert that the output of $\pi_{i,j}^k$ is $(z_{i,j}^k, v_k^{(i,j)})$ for some $z_{i,j}^k, v_k^{(i,j)}$.
     - If both (i, k) and (j, k) are contained in $G_k^{(m)}$, then output $z_{i,j}^k$.
     - Else if $\exists m', m'' \in \{i, j\}$ with $m' \neq m''$ such that $(m', k) \in G_k^{(m)}$, then
       * If $c_{m,m'}^{(k)} = 1$, output $z_{i,j}^k$.
       * Else output $z_{i,j}^k \oplus \alpha_m(s_{m,m''}^{(m')} \oplus s_{m,m''}^{(k)})$.
   - Else output $z_{i,j}^k$.

E.3 Proof of Theorem 4

In this section, we provide a sketch of the simulation and its analysis.

Simulation sketch. Let P_q denote the corrupt party. Acting as the honest parties, the simulator first sends random shares, random MAC values and random keys to corrupt P_q. Next, acting as the honest parties, the simulator receives shares and MAC values and keys from the corrupt party. Using these values the simulator computes $\{w_{p,q}\}_{p\in T_q}$, and then invokes $\mathsf{SimExtract}_q(\{w_{p,q}\}_{p\in T_q})$ (described in the previous section), and obtains the effective input s'_q. The simulator submits this input to the trusted party and obtains the output z'_q from the trusted party. Using this output, the simulator next invokes the PSM simulator $\mathcal{S}_\pi^{\text{trans}}$ for each of the PSM executions π from which P_q obtains output. To supply the inputs to $\mathcal{S}_\pi^{\text{trans}}$, the simulator takes advantage of the fact that f is a linear function, and that given an evaluation of linear function at a specific point s'_q, it is possible to obtain an evaluation at a different point s''_q. Recall that the value obtained from the trusted party, i.e., z'_q corresponds to the evaluation of f on corrupt input s'_q and the honest inputs $\{s_p\}_{p\in T_q}$. Now to obtain an evaluation of f on a different corrupt input s''_q and the same set of honest inputs $\{s_p\}_{p\in T_q}$, we just compute $z'_q \oplus \alpha_q(s'_q \oplus s''_q)$. This has the effect of canceling out $\alpha_q s'_q$ from z'_q and instead adding $\alpha_q s''_q$, thereby resulting in an evaluation on the desired set of points. Given this we claim that the simulator knows all the values that it needs to invoke $\mathcal{S}_\pi^{\text{trans}}$. To see why, first observe that the output of a PSM execution delivering output to P_q can never be of the form (w_i, w_j). This is because honest parties P_i, P_j will supply consistent CNF shares of each honest party's input. Let $k \in [4] \setminus \{q,i,j\}$. Recall that P_q supplied CNF shares $s_{q,j}^{(i)}, s_{q,k}^{(i)}$ to P_i and $s_{q,i}^{(j)}, s_{q,k}^{(j)}$ to P_j. It follows from the correctness property of the PSM protocol $\pi_{i,j}$ that if $s_{q,k}^{(i)} \neq s_{q,k}^{(j)}$, then the output of $\pi_{i,j}$ will be $\perp$. Now we only need to handle the case when the output of $\pi_{i,j}^q$ is of the form $(z_{i,j}^q, v_q^{(i,j)})$, i.e., in this case $s_{q,k}^{(i)} = s_{q,k}^{(j)}$. Thus, in this case the value $z_{i,j}^q$ would have been computed using the corrupt input $s''_q = s_{q,i}^{(j)} \oplus s_{q,j}^{(i)} \oplus s_{q,k}^{(i)}$. We just showed that the simulator can compute $z_{i,j}^q$ using the value z'_q obtained from the trusted party, the extracted input s'_q and the corrupt input s''_q used to compute the value $z_{i,j}^q$. Next, by inspection of the subroutine $\mathsf{LinRecView}_q^{(i,j)}$ which constructs the set $v_q^{(i,j)}$ it should be clear that the values contained in the set $v_q^{(i,j)}$ correspond to values sent by P_q in the first round of the protocol and to random additive shares sent by the simulator to P_q on behalf of the honest parties. Thus, we conclude that the simulator is able to successfully invoke the PSM simulator $\mathcal{S}_{\pi_{i,j}^q}^{\text{trans}}$ for each PSM protocol $\pi_{i,j}^q$ that delivers output to P_q. This concludes the description of the simulator.[2]

[2] Note that in the simulation we do not make use of the PSM simulator $\mathcal{S}_{\pi_{q,p}}^{\text{ext}}$ (guaranteed by the robustness property of the PSM protocol) for PSM protocols where P_q acts as a client. This is because we are concerned with full security and thus the simulation procedure must not depend on P_q's second round messages which for instance may not even be available in case P_q aborts without sending round 2 messages.

Analysis sketch. Since the corrupt party never obtains both MACs and keys for the same value, the values obtained from the simulator in the first round are indistinguishable from those obtained in the real execution. The messages that P_q receives in round 2 correspond to the PSM executions. In the simulation, these are messages that were generated by the PSM simulator $\mathcal{S}_\pi^{\text{trans}}$. In the description of the simulation above we saw how the simulator is able to compute the outputs that P_q will receive from each of the PSM executions. Given this and the privacy property of the PSM protocol, it follows that the view of the adversary in the ideal execution is indistinguishable from the real execution. More formally, in the analysis, we will consider a hybrid execution which is exactly the same as the real execution except the round 2 messages corresponding to the PSM protocols are generated by the PSM simulator $\mathcal{S}_\pi^{\text{trans}}$. From above it is obvious that the joint distribution of the view of the adversary and the honest outputs in the real execution is indistinguishable from the joint distribution of the view of the adversary and the honest outputs in the hybrid execution. Thus it remains to be shown that the hybrid execution is indistinguishable from the ideal execution.

The crux of the proof lies in showing that in the hybrid execution the output of the honest parties is generated using the honest inputs and the corrupt input s'_m extracted by the simulator, i.e., using the procedure $\mathsf{SimExtract}_q$ (since this is exactly how the honest outputs are generated in the ideal execution). To show this, we follow the case analysis used in the procedure $\mathsf{SimExtract}_q$. In the following we will focus on how (honest) party P_k computes its output in the hybrid execution. Let i, j be distinct indices in $[4] \setminus \{q, k\}$. We first consider the identifiable cases.

- *Identifiable triple-edge case.* In this case, note that the corrupt party P_q supplies shares that are pairwise inconsistent. That is every pair of honest parties holds inconsistent CNF shares of the corrupt party's input. We first claim that the inconsistency graph $G_k^{(q)}$ constructed by P_k will contain all three edges. Clearly (i, j) belongs to $G_k^{(q)}$ precisely because they hold inconsistent CNF shares of the corrupt party's input, i.e., $s_{q,k}^{(i)} \neq s_{q,k}^{(j)}$, and so the PSM execution $\pi_{i,j}^k$ will output (w_i, w_j). Now observe that w_i (resp. w_j) contains value $s_{q,j}^{(i)} \neq s_{q,j}^{(k)}$ (resp. $s_{q,i}^{(j)} \neq s_{q,i}^{(k)}$) since we are in the case where P_q supplied inconsistent CNF shares to every pair of honest parties. Thus, in Step 1 of the output computation procedure, P_k will add both (i, k) as well as (j, k) to the inconsistency graph $G_k^{(q)}$.
  Next, we claim that for every $p \in T_q$, the inconsistency graph $G_k^{(p)}$ does not contain all 3 edges. Consider $p = i$, and the inconsistency graph $G_k^{(i)}$. We claim that (j, k) is not an edge in $G_k^{(i)}$. Since P_i is honest, clearly both P_j and P_k hold consistent CNF shares, i.e., $s_{i,q}^{(j)} = s_{i,q}^{(k)}$. Thus immaterial of the output of $\pi_{q,j}^k$, the edge (j, k) will not be added to $G_k^{(i)}$.
  Given the above, now in Step 2 of the output computation procedure, P_k will set $m = q$. As we saw earlier, the output of the $\pi_{i,j}^q$ equals (w_i, w_j), and so the assertion passes. By inspection it follows that $\mathsf{LinRecInput}$ will

reconstruct the honest inputs correctly. Since in Step 2, the effective corrupt input used is 0, we conclude that in the identifiable triple-edge case, the hybrid execution is indistinguishable from the ideal execution.

– *Identifiable single-edge case.* Recall that for every $p \in T_q$, the inconsistency graph $G_k^{(p)}$ does not contain all 3 edges. First, we claim that $G_k^{(q)}$ will contain exactly one edge. Observe that since we are in the (identifiable) single-edge case there must be two pairs of honest parties that hold consistent shares. Let P_i, P_j hold consistent CNF shares of P_q's input, i.e., $s_{q,k}^{(i)} = s_{q,k}^{(j)}$. In this case, it is easy to see that $\pi_{i,j}^k$ will not output (w_i, w_j), and therefore edge (i, j) is not added to $G_k^{(q)}$. On the other hand suppose P_i, P_k hold consistent CNF shares of P_q's input, i.e., $s_{q,j}^{(i)} = s_{q,j}^{(k)}$. In this case by inspection it follows that (i, k) will not be an edge in $G_k^{(q)}$. That is, in either case, we have shown that if parties hold consistent CNF shares then they do not have an edge between them in $G_k^{(q)}$. On the other hand and as we saw earlier, if parties hold inconsistent CNF shares then there is an edge between them in $G_k^{(q)}$. Therefore, in the output computation procedure, honest P_k will skip Step 2 and go to Step 3 where the votes are computed for the single-edge.

  Now we claim that for every $p \in T_q$, if the inconsistency graph $G_k^{(p)}$ contains a single-edge then $c_{p,q}^{(k)} = 0$, i.e., P_p does not support P_q. (Note that it is obvious that q is one endpoint of this edge.) The argument is identical to the argument in the analysis of our VSS scheme, in that to get $c_{p,q}^{(k)} = 1$, party P_q has to forge an information-theoretic MAC on a value different from the one distributed by P_p; it can do so only with negligible probability. For the sake of clarity, we do not repeat the argument here, but mainly note that except with negligible probability, all parties agree on the outcome of all voting procedures. (See proof of Theorem 3 for more details.) In particular, this allows us to conclude party P_k will execute Step 4 of the output computation procedure with $m = q$.

  Now we have two cases to handle depending on whether k is a part of this single edge.

  - Suppose k is a part of the single edge. Denote this edge by (m'', k). Let $P_{m'}$ denote the honest party that is not part of the edge in $G_k^{(q)}$. First we claim that the output of $\pi_{m',m''}^k$ equals $(z_{m',m''}^k, v_k^{(m',m'')})$. This is because $P_{m'}, P_{m''}$ are both honest and hold consistent CNF shares of P_q's input (note: the single edge is (m'', k)). Now it is easy to see that the value $z_{m',m''}^k$ is computed using corrupt input $s_{q,m'}^{(m'')} \oplus s_{q,m''}^{(m')} \oplus s_{q,k}^{(m')}$. The output computation procedure then corrects this by xor-ing with $s_{q,m'}^{(k)}$ thereby effectively changing the corrupt input to $s_{q,m'}^{(m'')} \oplus s_{q,m''}^{(m')} \oplus s_{q,k}^{(m')} \oplus s_{q,m'}^{(k)}$, i.e., xor of all unique CNF shares (including the inconsistent ones). Since this is exactly the effective input extracted in the procedure $\mathsf{SimExtract}_q$, we conclude that the hybrid execution is indistinguishable from the real execution in this case.

- Suppose k is not part of the single edge, i.e., (i,j) is the single edge. Then clearly the output of $\pi^k_{i,j}$ equals (w_i, w_j) since they hold inconsistent shares (and that is precisely why there is a single edge between them), and thus the assertion succeeds. It then follows by simple inspection of Step 4 that the output is computed exactly as in the ideal execution.

This concludes the analysis of the identifiable single-edge case.

Next, we move to the resolvable cases. Before we begin, we observe that in these cases, the output computation procedure does not exit before Step 6. This is because recall that the inconsistency graphs $G_k^{(p)}$ for $p \in T_q$ neither contains 3 edges nor contains an identifiable single-edge.

– *Resolvable single-edge case.* In this case, it can be verified that the accusation graph contains at least one edge say $(q, \tilde{m})$ where $c^{(k)}_{q,\tilde{m}} = 0$. Note it is possible that $\tilde{m} = k$. In any case the output decision process terminates in either Step 7 or Step 8.
  - Suppose $\tilde{m} = k$. Then it is clear that the output decision procedure terminates in Step 8 (since obviously A_k contains at least one edge, and this edge is of the form (q,k)). Since we are in the resolvable single-edge case involving k as one of the endpoints of this edge, it is clear that P_i and P_j for $i,j \in T_{q,k}$ hold consistent CNF shares of the corrupt input. Therefore, the output of $\pi^k_{i,j}$ will not be of the form (w_i, w_j). In fact the output will be of the form $(z^k_{i,j}, v_k^{(i,j)})$. Again since we are in the (resolvable) single-edge case, at most one of (i,k), (j,k) is contained in $G_k^{(q)}$. Let $(m', k) \in G_k^{(q)}$ for $m' \in \{i,j\}$. Obviously $c^{(k)}_{q,m'} = 1$ (that is why it is a resolvable case), and so the output computation procedure terminates with output $z^k_{i,j}$. Now it remains to be shown that the output in the ideal execution also equals $z^k_{i,j}$.
    To show this, first let us see what inputs are used to compute output $z^k_{i,j}$ in the hybrid execution. Since P_i and P_j are honest parties, inside the execution $\pi^k_{i,j}$ the honest inputs are reconstructed using consistent CNF shares of honest inputs possessed by P_i and P_j. Likewise the corrupt input that is used to compute $z^k_{i,j}$ would be $s^{(j)}_{q,i} \oplus s^{(i)}_{q,j} \oplus s^{(i)}_{q,k}$, i.e., using consistent CNF shares possessed by P_i and P_j. Indeed this are exactly the inputs used to compute the output in the ideal execution in this case.
  - Suppose $\tilde{m} \neq k$, say $\tilde{m} = i$. First observe that in the ideal execution, the output is computed using honest inputs and the corrupt input that is reconstructed using consistent shares possessed by P_j and P_k. Thus it suffices to show the same in the hybrid execution. We split into two cases depending on whether (q,i) is the only edge in A_k or not.
    - Suppose (q,i) is the only edge in A_k. Then the output computation procedure terminates in Step 7.
      If for some $m' \in \{q,i\}$, the protocol $\pi^k_{m',j}$ outputs $(w_{m'}, w_j)$, then it is clear from the protocol description that the final output is computed exactly as in the ideal execution (i.e., using consistent CNF shares

possessed by P_j and P_k). Else note that the output of $\pi^k_{i,j}$ must be $(z^k_{i,j}, v^{(i,j)}_k)$ since (i,j) is not an edge in $G^{(q)}_k$ (recall we are in the resolvable single-edge case where this edge is (i,k)).

Next note that $v^{(i,j)}_k$ and v_k agree on all CNF shares received from honest parties. Clearly they do not agree on $s_{q,j}$ (which is precisely why (i,k) is an edge). Since $(j,k) \notin G^{(q)}_k$, it follows that $v^{(i,j)}_k$ and v_k also agree on CNF share $s_{q,i}$. Thus, the assertion in Step 7 holds. Note that the output $z^k_{i,j}$ is computed using corrupt input $s^{(j)}_{q,i} \oplus s^{(j)}_{q,k} \oplus s^{(i)}_{q,j}$. What we need is the output to be computed using extracted input $s^{(j)}_{q,i} \oplus s^{(j)}_{q,k} \oplus s^{(k)}_{q,j}$. This correction step is exactly what is performed in Step 7 when the assertion holds for $m' = i$ and $m'' = q$.

However the assertion in Step 7 may also hold in the reverse direction, i.e., for $m' = q$ and $m'' = i$. Fortunately, it can be verified that even in this case, the final output (obtained after correction in Step 7) is computed using honest inputs as well as the extracted corrupt input reconstructed from consistent CNF shares possessed by honest P_j and P_k.

  * Suppose A_k contains other edges besides (q,i). In this case, Step 8 is executed. First, observe that no two honest parties are connected by an edge in A_k (see Step 5). Next it can be verified that (q,k) will not be an edge in A_k in this case (i.e., in the single-edge case where P_q supports P_k). Thus the only other option that is left is that (q,j) also belongs to A_k. In this case, we see that if $\pi^k_{i,j}$ outputs (w_i, w_j), then the procedure $\mathsf{SimExtract}_q$ is invoked on values received by honest parties from P_q in round 1, to compute the corrupt input. Therefore we are assured that the hybrid execution is indistinguishable from the ideal execution.

    On the other hand if the output of $\pi^k_{i,j}$ is not (w_i, w_j), then it must indeed be $(z^k_{i,j}, v^{(i,j)}_k)$. Then since we are in the single-edge case with P_q supporting P_k, Step 8 terminates with output $z^k_{i,j} \oplus \alpha_q(s^{(i)}_{q,j} \oplus s^{(k)}_{q,j})$. This is perfect since $z^k_{i,j}$ is computed using corrupt input $s^{(j)}_{q,i} \oplus s^{(j)}_{q,k} \oplus s^{(i)}_{q,j}$. What we need is the output to be computed using extracted input $s^{(j)}_{q,i} \oplus s^{(j)}_{q,k} \oplus s^{(k)}_{q,j}$, and this is exactly what the correction in Step 8 does.

– *Resolvable double-edge case.* In this case, it can be verified that the accusation graph contains at least one edge say $(q, \tilde{m})$. Note it is possible that $\tilde{m} = k$. In any case the output decision process terminates in either Step 7 or Step 8.

  - Suppose $\tilde{m} = k$, i.e., the two edges in $G^{(q)}_k$ are (i,k) and (j,k). Then it is clear that the output decision procedure terminates in Step 8 (since obviously A_k contains at least one edge, and this edge is of the form (q,k)). Since we are in the resolvable double-edge case involving k as one of the endpoints of both edges, it is clear that P_i and P_j for $i,j \in T_{q,k}$ hold consistent CNF shares of the corrupt input. Therefore, the output of $\pi^k_{i,j}$ will not be of the form (w_i, w_j). In fact the output will be of the form $(z^k_{i,j}, v_k^{(i,j)})$. Also both (i,k), (j,k) are contained in $G_k^{(q)}$, and so the output computation procedure terminates with output $z^k_{i,j}$. Let us first see what inputs are used to compute output $z^k_{i,j}$ in the hybrid execution. Since P_i and P_j are honest parties, inside the execution $\pi^k_{i,j}$ the honest inputs are reconstructed using consistent CNF shares of honest inputs possessed by P_i and P_j. Likewise the corrupt input that is used to compute $z^k_{i,j}$ would be $s^{(j)}_{q,i} \oplus s^{(i)}_{q,j} \oplus s^{(i)}_{q,k}$, i.e., using consistent CNF shares possessed by P_i and P_j. Indeed this are exactly the inputs used to compute the output in the ideal execution in this case, so we have that the ideal execution is indistinguishable from the hybrid execution.

- Suppose $\tilde{m} \neq k$, say $\tilde{m} = i$, i.e., the two edges in $G_k^{(q)}$ are (i,k) and (i,j). First observe that in the ideal execution, the output is computed using honest inputs and the corrupt input that is reconstructed using consistent shares possessed by P_j and P_k. Thus it suffices to show the same in the hybrid execution. We split into two cases depending on whether (q,i) is the only edge in A_k or not.
  - ∗ Suppose (q,i) is the only edge in A_k. Then the output computation procedure terminates in Step 7.
    In fact, $\pi^k_{i,j}$ will output (w_i, w_j) since $(i,j) \in G_k^{(q)}$. Thus, it is clear from the protocol description that the final output is computed exactly as in the ideal execution (i.e., using consistent CNF shares possessed by P_j and P_k).
  - ∗ Suppose A_k contains other edges besides (q,i). In this case, Step 8 is executed. First, observe that no two honest parties are connected by an edge in A_k (see Step 5). Next it can be verified that (q,k) will not be an edge in A_k in this case (i.e., in the double-edge case with edges (i,j) and (i,k)). Thus the only other option that is left is that (q,j) also belongs to A_k. Since $\pi^k_{i,j}$ outputs (w_i, w_j), the procedure $\mathsf{SimExtract}_q$ is invoked on values received by honest parties from P_q in round 1, to compute the corrupt input. Therefore we are assured that the hybrid execution is indistinguishable from the ideal execution.

– *Resolvable zero-edge case.* In this case, all three honest parties hold consistent CNF shares of the corrupt party's input. We split the analysis into three cases depending on the structure of A_k.

- Suppose A_k contains no edges. Clearly, $\pi^k_{i,j}$ outputs $(z^k_{i,j}, v_k^{(i,j)})$ since both P_i, P_j are honest and hold consistent CNF shares for all parties. In this case simply accepting $z^k_{i,j}$ as in Step 6 guarantees that the hybrid execution is indistinguishable from the ideal execution.
- Suppose A_k contains exactly one edge (q,i). If for some $m' \in \{q,i\}$, the protocol $\pi^k_{m',j}$ outputs $(w_{m'}, w_j)$, then it is clear from the protocol description that the final output is computed exactly as in the ideal execution (i.e., using consistent CNF shares possessed by P_j and P_k).

Else note that the output of $\pi_{i,j}^k$ must be $(z_{i,j}^k, v_k^{(i,j)})$ since (i,j) is not an edge in $G_k^{(q)}$ (recall we are in the resolvable zero-edge case).

Next note that $v_k^{(i,j)}$ and v_k agree on all CNF shares received from honest parties as well as the corrupt party P_q. Thus, the assertion in Step 7 holds. Note that the output $z_{i,j}^k$ is computed using corrupt input reconstructed from consistent shares held by P_i and P_j.

However the assertion in Step 7 may also hold in the reverse direction, i.e., for $m' = q$ and $m'' = i$. Fortunately, it can be verified that even in this case, the final output (obtained after correction in Step 7) is computed using honest inputs as well as the extracted corrupt input reconstructed from consistent CNF shares possessed by honest P_j and P_k.

- Suppose A_k contains the edge (q,k) or contains two edges (q,i) and (q,j). Actually one can verify that (q,k) can never be part of A_k in this case since there are no edges in $G_k^{(q)}$. For the rest of the analysis assume that A_k contains two edges (q,i) and (q,j) (these could be added after inspecting the structure of say $G_k^{(i)}, G_k^{(j)}$).

  In this case, Step 8 is executed. Obviously the output of $\pi_{i,j}^k$ is not (w_i, w_j); it must be $(z_{i,j}^k, v_k^{(i,j)})$. In this case, it is easy to see that the protocol terminates with output $z_{i,j}^k$. This is indeed the output in the ideal execution as well. Thus we conclude that the hybrid execution is indistinguishable from the ideal execution.

This concludes the analysis sketch.

F More Details on 2-Round 4-Party Computationally Secure Protocol

F.1 Protocol Description

Subroutines. In order to simplify the description of the protocol, we use subroutines $\mathsf{RecInput}$ and $\mathsf{RecView}$. We start by describing $\mathsf{RecInput}$ which is used to reconstruct inputs from commitments and (possibly inconsistent) shares of decommitments. Recall that for $i,j \in [4]$, $T_{i,j}$ denotes the set $[4] \setminus \{i,j\}$.

<u>SUBROUTINE $\mathsf{RecInput}_k^{(i,j)}(v_k^{(i)}, v_k^{(j)})$</u>

- Inputs: $v_k^{(i)} = (c_k^{(i)}, \{\gamma_{k,t}^{(i)}\}_{t \in T_{i,k}})$ and $v_k^{(j)} = (c_k^{(j)}, \{\gamma_{k,t}^{(j)}\}_{t \in T_{j,k}})$.
- If $c_k^{(i)} \neq c_k^{(j)}$, output $\perp$ and terminate. Else, set $c'_k = c_k^{(i)}$.
- Find γ'_k that is a valid decommitment for c'_k s.t. $\gamma'_k = \bigoplus_{t \in T_k} \gamma'_{k,t}$ with $\gamma'_{k,t} \in \{\gamma_{k,t}^{(i)}\}_{t \in T_{i,k}} \cup \{\gamma_{k,t}^{(j)}\}_{t \in T_{j,k}}$.
- If no such γ'_k exists, then output $\perp$. Else, let $\gamma'_k = (s'_k, *)$, and output s'_k.

Remark. The third step of $\mathsf{RecInput}$ tries at most 2 possibilities for valid decommitment. Also if more than one valid decommitment exists, the subroutine returns the lexicographically smallest decommitment.

Next, we describe subroutine $\mathsf{RecView}$ which is used to reconstruct a view of the referee that is consistent with the views of the PSM clients.

<u>SUBROUTINE $\mathsf{RecView}_k^{(i,j)}(v_i, v_j)$</u>

- Inputs: $v_i = \{v_m^{(i)} = (c_m^{(i)}, \{\gamma_{m,t}^{(i)}\}_{t\in T_{i,m}})\}_{m\in[4]}$ and $v_j = \{v_m^{(j)} = (c_m^{(j)}, \{\gamma_{m,t}^{(j)}\}_{t\in T_{j,m}})\}_{m\in[4]}$.
- If $\exists m \in \{i,j,k\}$ such that $c_m^{(i)} \neq c_m^{(j)}$, output $\perp$ and terminate.
- If $\gamma_{i,k}^{(i)} \neq \gamma_{i,k}^{(j)}$ or if $\gamma_{j,k}^{(i)} \neq \gamma_{j,k}^{(j)}$, output $\perp$ and terminate.
- Output $v_k^{(i,j)} = \{\gamma_{m,t}^{(m)}\}_{m\in\{i,j\},t\in T_{m,k}}$.

The protocol for 4-party secure computation is described in Figure 3. In Appendix F.2 we prove:

Protocol. Let $(\mathsf{Com}, \mathsf{Dec})$ be a non-interactive commitment scheme.

Round 1. For each $i \in [4]$: Let s_i denote P_i's input. P_i chooses random ω_i and computes $c_i = \mathsf{Com}(s_i; \omega_i)$, and sets $\gamma_i = (s_i, \omega_i)$. Let $\{\gamma_{i,j}\}_{j\in[4]\setminus\{i\}}$ denote the shares corresponding to a 1-private 3-party CNF sharing of γ_i. P_i sends to each P_j, the values $\{\gamma_{i,t}^{(j)} = \gamma_{i,t}\}_{t\in T_{i,j}}$, and broadcasts c_i to all parties. In addition, P_i also exchanges randomness with each P_j for a 2-client PSM protocol described below. For $i, k \in [4]$, let $v_k^{(i)}$ denote $(c_k^{(i)}, \{\gamma_{k,t}^{(i)}\}_{t\in T_{i,k}})$.

Round 2. Each pair of parties (P_i, P_j) runs the following PSM protocol $\pi_{i,j}^{\ell}$ that delivers output to P_ℓ:

- Inputs: $v_i = \{v_k^{(i)}\}_{k\in[4]}$ from P_i, and $v_j = \{v_k^{(j)}\}_{k\in[4]}$ from P_j.
- For all $k \in [4]$, compute $s'_k = \mathsf{RecInput}_k^{(i,j)}(v_k^{(i)}, v_k^{(j)})$.
- If $\exists k \in [4]$ such that $s'_k = \perp$, output $\perp$ if $s'_\ell = \perp$, else output (v_i, v_j).
- Else, output $(z_{i,j}, v_\ell^{(i,j)})$, where $z_{i,j} = f(s'_1, \ldots, s'_4)$ and $v_\ell^{(i,j)} = \mathsf{RecView}_\ell^{(i,j)}(v_i, v_j)$.

Output Computation. Each P_k reconstructs its output as follows.

1. If $\exists i, j \in T_k$ such that $\pi_{i,j}^k$ outputs $(z_{i,j}, \{\gamma_{m,t}^{(k)}\}_{m\in\{i,j\},t\in T_{m,k}})$, then output $z_{i,j}$.
2. Else if $\exists i, j \in T_k$ such that $\pi_{i,j}^k$ outputs (v_i, v_j), and for $s_m^{(m_1,m_2)} \triangleq \mathsf{RecInput}_m^{(m_1,m_2)}(v_m^{(m_1)}, v_m^{(m_2)})$, it holds that $\forall m \in \{i,j,k\}$, $s_m^{(i,k)} = s_m^{(j,k)} = s_m^{(i,j)} \neq \perp$, then (a) $\forall m \in \{i,j,k\}$, set $s''_m = s_m^{(i,j)}$, and (b) for $\ell \notin \{i,j,k\}$, set $s''_\ell = 0$, and (c) if $\exists m_1, m_2 \in \{i,j,k\}$ such that $s_\ell^{(m_1,m_2)} = \mathsf{RecInput}_\ell^{(m_1,m_2)}(v_\ell^{(m_1)}, v_\ell^{(m_2)}) \neq \perp$, then set $s''_\ell = s_\ell^{(m_1,m_2)}$, and (d) output $f(s''_1, \ldots, s''_4)$.

Fig. 3. 2-round 4-party computationally secure protocol.

Lemma 2. *Assuming the existence of one-way permutations (alternatively, one-to-one one-way functions), there exists a 2-round 4-party computationally secure protocol for secure function evaluation that tolerates a single malicious party and uses broadcast in the first round only.*

F.2 Proof of Lemma 2

We first provide an informal overview of the simulator.

Overview. The simulator begins by sending commitments on 0 to the corrupt party on behalf of the honest parties. Then, it chooses random CNF shares and sends these to the corrupt party as decommitment shares received from honest parties. At this stage, the simulator is ready to receive the decommitment shares along with the broadcasted commitment from the corrupt party. Then, it checks if the joint view of the honest parties contains a *unique* valid decommitment to the commitment of the corrupt party, and in this case it extracts the input of the corrupt party from the valid decommitment. If there is more than one valid decommitment, then the corrupt party has violated the binding property of the commitment (which can happen only with negligible probability due to security of the commitment scheme), and in this case the simulator outputs fail and terminates the simulation. Else if there is no valid decommitment that can be reconstructed from the decommmitment shares in the joint view of honest parties, then the simulator sets the input of the corrupt party to 0. At the end of the first round, the simulator sends the extracted input to the trusted party and receives back output from the trusted party. If the joint view of the honest parties did not contain a valid decommitment to the commitment of the corrupt party, then the simulator discards the output received from the trusted party and sets $\perp$ as the final output of the protocol.
In the second round, the simulator prepares the PSM client messages to send to the corrupt party. To generate these messages, the simulator first computes the output that each of these PSM instances need to deliver, and then it invokes the PSM simulator (denoted $\mathcal{S}_\pi^{\text{trans}}$ for PSM instance π) to obtain transcripts of the PSM protocols by providing it the corresponding output. The output of the PSM instances is determined based on whether the joint view of the two PSM clients contain a valid decommitment to the commitment broadcasted by the corrupt party. If this is the case, then the output of the PSM is set to the output received from the trusted party. In the other case, the output of the PSM is set to $\perp$. Then, the simulator receives PSM messages from the corrupt party, and runs $\mathcal{S}_\pi^{\text{ext}}$ to extract the PSM input, which corresponds to its first round view, supplied by the adversary. If there exists an honest party such that the joint view of this honest party and the corrupt party contains a valid decommitment to the commitment broadcasted by the corrupt party that is different from the simulator's extracted input, then the simulator outputs fail_1 and terminates the simulation. Else it outputs whatever the adversary outputs and terminates the simulation. This concludes the informal description of the simulator.

We formally describe the simulation for corrupt party, say P_ℓ below.

Simulating corrupt P_ℓ. For each $m \in T_\ell$, the simulator acting as P_m does the following:

- Choose random ω_m and send $c_m = \mathsf{Com}(0; \omega_m)$ over the broadcast channel.
- Send random $\gamma_{m,t}$ for each $t \in [4] \setminus \{m, \ell\}$ to P_ℓ over point-to-point channels.
- Send PSM randomness $r^{\mathrm{psm}}_{m,\ell,t}$ to P_ℓ over point-to-point channels for $t \in T_{m,\ell}$ if $m < \ell$.
- Receive from P_ℓ values $\{\gamma^{(m)}_{\ell,t}\}_{t\in T_{m,\ell}}$ over point-to-point channels and c_ℓ over the broadcast channel.
- Receive from P_ℓ PSM randomness $r^{\mathrm{psm}}_{\ell,m,t}$ for $t \in T_{m,\ell}$ if $m > \ell$.

Next, the simulator extracts P_ℓ's input in the following way. For each $m \in T_\ell$, set $v^{(m)}_\ell = (c_\ell, \{\gamma^{(m)}_{\ell,t}\}_{t\in T_{m,\ell}})$, and run the following subroutine.

Subroutine $\mathsf{Extract}_\ell(\{v^{(m)}_\ell\}_{m\in T_\ell})$

- For distinct $m_1, m_2 \in T_\ell$, compute $s^{(m_1,m_2)}_\ell = \mathsf{RecInput}^{(m_1,m_2)}_\ell(v^{(m_1)}_\ell, v^{(m_2)}_\ell)$.
- Initialize $S_\ell = \emptyset$. For all $m_1, m_2 \in T_\ell$ add $s^{(m_1,m_2)}_\ell$ to S_ℓ if $s^{(m_1,m_2)}_\ell \neq \perp$.
- If $|S_\ell| > 1$, then output (fail, S_ℓ).
- Else if $|S_\ell| = 0$, then output $(\mathsf{bad}, 0)$.
- Else, let s_ℓ denote the unique element in S_ℓ, and output (good, s_ℓ).

Let (code, y_1) denote the output of the $\mathsf{Extract}$ subroutine. If $\mathsf{code} = \mathsf{fail}$, then the simulator outputs fail and terminates the simulation. Else, the simulator sends y_1 to the trusted party. Let z_ℓ denote the output received from the trusted party. In the next step, the simulator prepares to send the second round messages to P_ℓ by executing the following for all pairs (m_1, m_2) with $m_1 < m_2$.

Subroutine $\mathsf{PsmTrans}^{(m_1,m_2)}_\ell(v^{(m_1)}_\ell, v^{(m_2)}_\ell, \{\gamma_{m,t}\}_{m\in\{m_1,m_2\},t\in T_{m,\ell}}, z_\ell)$

- Compute $s^{(m_1,m_2)}_\ell = \mathsf{RecInput}^{(m_1,m_2)}_\ell(v^{(m_1)}_\ell, v^{(m_2)}_\ell)$.
- Set $y_{m_1,m_2} = \perp$ if $s^{(m_1,m_2)}_\ell = \perp$ else set $y_{m_1,m_2} = (z_\ell, \{\gamma_{m,t}\}_{m\in\{m_1,m_2\},t\in T_{m,\ell}})$.
- Invoke PSM simulator $\mathcal{S}^{\mathrm{trans}}_{\pi^\ell_{m_1,m_2}}(1^\kappa, y_{m_1,m_2})$ to obtain transcript $\tau^{(m_1,m_2)}_{m_1}, \tau^{(m_1,m_2)}_{m_2}$.
- For all $m \in \{m_1, m_2\}$, acting as P_m sends $\tau^{(m_1,m_2)}_m$ to P_ℓ over point-to-point channels.

For each $i \in T_\ell, k \in T_{\ell,i}$, $\mathcal{S}$ receives PSM messages $\tilde{\tau}^{(i,k)}_\ell$ from the adversary for execution $\pi^k_{\ell,i}$. (Recall that $r^{\mathrm{psm}}_{\ell,i,k}$ denotes the PSM randomness used in execution $\pi^k_{\ell,i}$. $\mathcal{S}$ then executes the following subroutine.

Subroutine $\mathsf{PsmExtract}_\ell(\{r^{\mathrm{psm}}_{\ell,i,k}, \tilde{\tau}^{(i,k)}_\ell\}_{i\in T_\ell,k\in T_{\ell,i}}, \{v^{(m)}_\ell\}_{m\in T_\ell})$

- For each $i \in T_\ell, k \in T_{i,\ell}$, invoke PSM simulator $\mathcal{S}^{\mathrm{ext}}_{\pi^k_{\ell,i}}(1^\kappa, r^{\mathrm{psm}}_{\ell,i,k}, \tilde{\tau}^{(i,k)}_\ell)$ to obtain output $\tilde{v}_{\ell,i,k} = \{\tilde{v}^{(\ell,i,k)}_m\}_{m\in[4]}$.
- If $\exists i \in T_\ell, k \in T_{\ell,i}$ such that $\tilde{v}_{\ell,i,k} = \perp$ (i.e., $\mathcal{S}^{\mathrm{ext}}_{\pi^k_{\ell,i}}$ failed), then output $\mathsf{psm\text{-}fail}$ and terminate.

– Initialize $\tilde{S}_\ell = \emptyset$. For each $i \in T_\ell, k \in T_{\ell,i}$, compute $\tilde{s}_\ell^{(i,k)} = \mathsf{RecInput}_\ell^{(\ell,i)}(\tilde{v}_\ell^{(\ell,i,k)}, v_\ell^{(i)})$, and add $\tilde{s}_\ell^{(i,k)}$ to $\tilde{S}_\ell$.

If the output is psm-fail, then $\mathcal{S}$ outputs psm-fail and terminates. Else if $\tilde{S}_\ell \not\subseteq \{y_1, \perp\}$, then $\mathcal{S}$ outputs fail_1 and terminates the simulation. Else, $\mathcal{S}$ outputs whatever the adversary outputs and terminates the simulation.

Analysis. We construct a sequence of hybrids starting with the real execution and ending with the simulated execution and prove that each hybrid is indistinguishable from the next.

Hybrid H_0. This is identical to the real execution of the protocol. We can restate the above hybrid with the simulator as follows. We replace the real world adversary $\mathcal{A}$ with the ideal world adversary $\mathcal{S}$. The ideal adversary $\mathcal{S}$ starts by invoking a copy of $\mathcal{A}$ and running a simulated interaction of $\mathcal{A}$ and the honest parties. In this hybrid the simulator $\mathcal{S}$ holds the private inputs of the honest parties and generates messages on their behalf using the honest party strategies as specified by the protocol.

Hybrid H_1. In this hybrid we change how the simulator generates output of the honest parties. In particular, we let $\mathcal{S}$ extract the input of the corrupted party by running the Extract subroutine. Let (code, y_1) be the output of the Extract subroutine. If $\mathsf{code} = \mathsf{fail}$, then $\mathcal{S}$ output fail and terminates. Else if $\mathsf{code} \neq \mathsf{fail}$, then $\mathcal{S}$ uses y_1 as $\mathcal{A}$'s input, and computes output of the honest parties, say z_ℓ. Then $\mathcal{S}$ obtains the PSM messages from $\mathcal{A}$ and runs the subroutine PsmExtract as described above. If the output of PsmExtract is psm-fail, then $\mathcal{S}$ outputs psm-fail and terminates. Otherwise, let $\tilde{S}_\ell$ be the output of PsmExtract. If $\tilde{S}_\ell \not\subseteq \{y_1, \perp\}$, then $\mathcal{S}$ outputs fail_1, and terminates the simulation. Else, $\mathcal{S}$ outputs whatever the adversary outputs and terminates the simulation.

First, we claim that the probability that $\mathcal{S}$ outputs psm-fail is negligible in κ. This follows directly from the security (more precisely, robustness property) of the PSM protocol π. Next, we claim that the probability that $\mathcal{S}$ outputs fail or fail_1 is negligible in κ. Indeed, this is the case, since an adversary that makes $\mathcal{S}$ output fail or fail_1 can be easily used to break the binding property of the commitment scheme. Since we use a secure commitment scheme, it follows that the probability that $\mathcal{S}$ outputs fail or fail_1 is negligible in κ. We continue the analysis conditioned on neither event happening.

Suppose $\mathsf{code} = \mathsf{good}$, then we argue that the output of honest parties in H_0 is identical to their output in H_1. Let $i, j \in T_\ell$ be the parties such that their joint view contains a valid decommitment for c_ℓ (i.e., the commitment broadcasted by $\mathcal{A}$ on behalf of P_ℓ). In this case, clearly, P_k with $k \notin \{i, j, \ell\}$ obtains z_ℓ as output of $\pi_{i,j}^k$ (i.e., exactly the output computed by $\mathcal{S}$ in H_1). Further, when $\tilde{S}_\ell = \{\tilde{s}_\ell^{(m)}\}_{m \in T_\ell} \subseteq \{y_1, \perp\}$, the output of $\pi_{\ell,i}^k$ is either $(z_\ell, \star)$ (when $\tilde{s}_i^{(m)} = y_1$), or (v_ℓ, v_i) (when $\tilde{s}_i^{(m)} = \perp$). Note that in either case, the output of P_k remains unchanged. It remains to be shown that each of P_i, P_j also obtain the same output. Below we analyse the output of P_j. (The analysis for P_i is identical *mutatis mutandis*.) Indeed if the joint view of P_k and P_i also contains a valid decommitment for c_ℓ, then P_j obtains as output from $\pi^j_{k,i}$ the value z_ℓ since $\mathsf{code} = \mathsf{good}$ and so the joint view of honest parties P_i, P_j, P_k contains a unique decommitment for c_ℓ. On the other hand, if the joint view of P_k and P_i does not contain a valid decommitment for c_ℓ, then the output of $\pi^j_{k,i}$ would be (v_k, v_i), and party P_j can reconstruct a valid decommitment from the joint view (v_i, v_j), and reconstruct input of corrupt P_ℓ as well as honest parties P_i, P_j, and P_k. Then, using these extracted inputs, P_j computes the output of the function. As before, $\mathsf{code} = \mathsf{good}$, and so the output value computed by P_j is the same as the one computed by P_k.

Now suppose $\mathsf{code} = \mathsf{bad}$. Note that the output of $\mathsf{Extract}$ is such that if $\mathsf{code} = \mathsf{bad}$, then $y_1 = 0$. Therefore, in H_1 the outputs are computed by substituting the value 0 for the corrupt party's input. We claim that the outputs of the honest parties in H_0 are computed in an identical manner. This is because, when $\mathsf{code} = \mathsf{bad}$, the joint view of all honest parties, say P_i, P_j, P_k does not contain a valid decommitment to the commitment broadcasted by the corrupt party. Consider an honest party P_k. We prove that the output of P_k is computed by substituting the corrupt party's input by 0. (The argument is identical for other honest parties P_i, P_j.) First, note that the output of the PSM protocol $\pi^k_{i,j}$ is (v_i, v_j) since for $s'_\ell = \mathsf{RecInput}^{(i,j)}_\ell$ it holds that $s'_\ell = \perp$. Next, consider $\pi^k_{\ell,i}$. (The analysis is identical for $\pi^k_{\ell,j}$.) If the output of $\pi^k_{\ell,i}$ is $\perp$, then the claim holds. Else, the output is either of the form $(z, v^{(i,j)}_\ell)$ or of the form (v_ℓ, v_i). In the first case, note that $s'_\ell = \mathsf{RecInput}^{(i,j)}_\ell \neq \perp$, i.e., the joint view of P_i and P_ℓ contains a valid decommitment for c_ℓ. The subroutine $\mathsf{RecView}^{(i,j)}_k$ recreates the view of P_k, in particular P_k's decommitment share consistent with P_i's decommitment share such that the shares together define the valid decommitment for c_ℓ. Since $\mathsf{code} = \mathsf{bad}$, the recreated share (and therefore the view) does not match with P_k's first round view, and therefore P_k rejects z as the final output. Recall that in the second case, the output is of the form (v_ℓ, v_i). In this case, the condition $s^{(i,k)}_\ell \neq \perp$ does not hold since $\mathsf{code} = \mathsf{bad}$ and so the joint view of P_k and P_i does not contain a valid decommitment for c_ℓ. In summary, execution $\pi^k_{\ell,i}$ is not used for generating P_k's output. On the other hand, the condition $\forall m \in \{i, j, k\}, s^{(i,k)}_m = s^{(j,k)}_m = s^{(i,j)}_m \neq \perp$ *does* hold for the pair (v_i, v_j), i.e., the output of $\pi^k_{i,j}$. In particular for every $m \in \{i, j, k\}$ the input s''_m is computed using the views of honest parties P_i, P_j, P_k, and further, for $\ell \notin \{i, j, k\}$, P_ℓ's input is substituted by 0. The output of the function computed on inputs derived as described above is then accepted by P_k.

Hybrid H_2. In this hybrid instead of generating the PSM messages on behalf of honest parties, $\mathcal{S}$ uses $\mathcal{S}^{\text{trans}}_\pi$ (the simulator for the underlying PSM protocol) to generate simulated messages. In particular, as in H_1 the simulator now extracts $\mathcal{A}$'s input and uses this along with the private inputs of the honest parties and the extracted input to compute the output z_ℓ. Then $\mathcal{S}$ computes the PSM messages that would be delivered to P_ℓ via the $\mathsf{PsmTrans}$ subroutine using messages that

it sent to/received from P_ℓ and the computed output z_ℓ. $\mathcal{S}$ sends these simulated PSM messages to $\mathcal{A}$ instead of the honest PSM messages.
Note that the output derived from messages output by $\mathsf{PsmTrans}_\ell^{(m_1,m_2)}$ is exactly the same as the output that $\mathcal{A}$ receives from $\pi^\ell_{m_1,m_2}$ in H_1. It then follows from (a straightforward hybrid argument involving) the security of the PSM protocol that the distribution of the messages received from $\mathcal{S}$ in H_2 is indistinguishable from the distribution of the messages received in H_1.

Hybrid H_3. In this hybrid we change how the simulator $\mathcal{S}$ generates the first round messages on behalf of the honest parties. In particular instead of committing to the inputs of honest parties $\mathcal{S}$ just sends commitments on zero strings of appropriate length.
Indistinguishability between hybrids H_2 and H_3 directly follows from (a straightforward hybrid argument involving) the hiding property of the commitment scheme.

Hybrid H_4. Observe that in hybrid H_3, $\mathcal{S}$ uses inputs of honest parties only to obtain the output of the computation. Instead, $\mathcal{S}$ can obtain the same value by sending extracted input of the adversary to the trusted party.
Note that hyrids H_3 and H_4 are identical. Also observe that hybrid H_4 is identical to the simulation strategy. This concludes the proof.

F.3 2-Round Computationally Secure 4-Party Computation Over Point-to-Point Channels

Our first observation is that parties use the broadcast channel in protocol described only to broadcast their commitments. Our strategy to get rid of broadcast is simple: we just let the parties send their commitments over point-to-point channels instead. This however introduces several subtle problems which we will need to address.

How to extract. We first design the simulation extraction procedure which will serve as the guiding light in the design of our protocol Although our procedure will be quite similar to the extraction procedure $\mathsf{Extract}_\ell$ (used in the previous subsection) in the case where a broadcast channel was available, here we need to take care of the obvious issue in that parties may hold inconsistent values for the commitment c_ℓ. To resolve this, we use the majority value among the commitments received by the honest parties. That is, we assume $\hat{c} = \mathsf{majority}(\{c_\ell^{(p)}\}_{p \in T_\ell})$ as the commmitment value that was broadcast. Then as before, we try to see whether a decommitment can be constructed using the (possibly inconsistent) CNF shares possessed by any pair of parties. If no such decommitment exists, then we extract the corrupt input as 0. Else, we use the decommitment to extract the corrupt input in the obvious way. This concludes the description of the extraction procedure.

Additional subroutines. Now to force parties to accept inputs that are computed only according to the extracted corrupt input, we need to design a new subroutine called $\mathsf{RecInputNoBC}_k^{(i,j)}$. As we will see, this subroutine is used only in the output computation step (while inside the PSM protocol the subroutine $\mathsf{RecInput}_k^{(i,j)}$ is executed as before). The main difference between $\mathsf{RecInput}$ and $\mathsf{RecInputNoBC}$ is that the latter takes an additional input to ascertain the majority value among the commitments possessed by various parties.

Subroutine $\mathsf{RecInputNoBC}_k^{(i,j)}(\tilde{c}, v_k^{(i)}, v_k^{(j)})$

- Inputs: $v_k^{(i)} = (c_k^{(i)}, \{\gamma_{k,t}^{(i)}\}_{t\in T_{i,k}})$ and $v_k^{(j)} = (c_k^{(j)}, \{\gamma_{k,t}^{(j)}\}_{t\in T_{j,k}})$.
- If $c_k^{(i)} \neq c_k^{(j)} \neq \tilde{c}_k \neq c_k^{(i)}$, output $\perp$ and terminate. Else, set $c'_k = \mathsf{majority}(c_k^{(i)}, c_k^{(j)}, \tilde{c})$.
- Find γ'_k that is a valid decommitment for c'_k s.t. $\gamma'_k = \bigoplus_{t\in T_k} \gamma'_{k,t}$ with $\gamma'_{k,t} \in \{\gamma_{k,t}^{(i)}\}_{t\in T_{i,k}} \cup \{\gamma_{k,t}^{(j)}\}_{t\in T_{j,k}}$.
- If no such γ'_k exists, then output $\perp$. Else, let $\gamma'_k = (s'_k, *)$, and output s'_k.

We also need to replace the $\mathsf{RecView}_k^{(i,j)}$ subroutine with the subroutine $\mathsf{RecViewNoBC}_k^{(i,j)}$ described below.

Subroutine $\mathsf{RecViewNoBC}_k^{(i,j)}(v_i, v_j)$

- Inputs: $v_i = \{v_m^{(i)} = (c_m^{(i)}, \{\gamma_{m,t}^{(i)}\}_{t\in T_{i,m}})\}_{m\in[4]}$ and $v_j = \{v_m^{(j)} = (c_m^{(j)}, \{\gamma_{m,t}^{(j)}\}_{t\in T_{j,m}})\}_{m\in[4]}$.
- If $\exists m \in \{i,j,k\}$ such that $c_m^{(i)} \neq c_m^{(j)}$, output $\perp$ and terminate.
- If $\gamma_{i,k}^{(i)} \neq \gamma_{i,k}^{(j)}$ or if $\gamma_{j,k}^{(i)} \neq \gamma_{j,k}^{(j)}$, output $\perp$ and terminate.
- Output $v_k^{(i,j)} = \{c_m^{(m)}, \{\gamma_{m,t}^{(m)}\}_{t\in T_{m,k}}\}_{m\in\{i,j\}}$.

The main difference between $\mathsf{RecView}_k^{(i,j)}$ and $\mathsf{RecViewNoBC}_k^{(i,j)}$ is that in the latter, the output also includes the commitments $c_i^{(i)}$ and $c_j^{(j)}$ as well. This is to prevent attacks in which a malicious party P_ℓ sends different commitment values to different parties, and there is no majority among these commitments. Then inside a PSM subprotocol where the corrupt party is a client, it supplies the commitment equal to commitment that it earlier sent to the other client thereby creating an illusion of majority (note inside such a PSM, $\mathsf{RecInput}_\ell$ will succeed, while for every other PSM in which both clients are honest $\mathsf{RecInput}_\ell$ will fail). By including the commitments as part of the output, we ensure that an honest party accepts the output of this PSM only if the commitment value that it possesses matches the commitment values present in the output of the PSM protocol. By doing so, we are ensured that there indeed exists a majority among the commitments distributed among the honest parties.

We now describe our protocol for 4-party secure computation over point-to-point channels. In the following, let T_i denote the set $[4] \setminus \{i\}$, and let $T_{i,j}$ denote the set $[4] \setminus \{i,j\}$.

Protocol. Let $(\mathsf{Com}, \mathsf{Dec})$ be a non-interactive commitment scheme.

Round 1. For each $i \in [4]$: Let s_i denote P_i's input. P_i chooses random ω_i and computes $c_i = \mathsf{Com}(s_i; \omega_i)$, and sets $\gamma_i = (s_i, \omega_i)$. Let $\{\gamma_{i,j}\}_{j \in [4] \setminus \{i\}}$ denote the shares corresponding to a 1-private 3-party CNF sharing of γ_i.

$\star$ Party P_i sends to each P_j, the values $\{\gamma_{i,t}^{(j)} = \gamma_{i,t}\}_{t \in T_{i,j}}$, and the commitment $c_i^{(j)} = c_i$ to all parties.

In addition, P_i also exchanges randomness with each P_j for a 2-client PSM protocol described below. For $i, k \in [4]$, let $v_k^{(i)}$ denote $(c_k^{(i)}, \{\gamma_{k,t}^{(i)}\}_{t \in T_{i,k}})$.

Round 2. Each pair of parties (P_i, P_j) runs the following PSM protocol $\pi_{i,j}^{\ell}$ that delivers output to P_ℓ:

- Inputs: $v_i = \{v_k^{(i)}\}_{k \in [4]}$ from P_i, and $v_j = \{v_k^{(j)}\}_{k \in [4]}$ from P_j.
- For all $k \in [4]$, compute $s'_k = \mathsf{RecInput}_k^{(i,j)}(v_k^{(i)}, v_k^{(j)})$.
- If $\exists k$ such that $s'_k = \bot$, output $\bot$ if $s'_\ell = \bot$, else output (v_i, v_j).
- Else, output $(z_{i,j}, v_\ell^{(i,j)})$, where $z_{i,j} = f(s'_1, \dots, s'_4)$ and $v_\ell^{(i,j)} = \mathsf{RecViewNoBC}_\ell^{(i,j)}(v_i, v_j)$.

Output Computation. Each P_k reconstructs its output as follows.

1. If $\exists i, j \in T_k$ such that $\pi_{i,j}^k$ outputs $(z_{i,j}, \{c_m^{(k)}, \{\gamma_{m,t}^{(k)}\}_{t \in T_{m,k}}\}_{m \in \{i,j\}})$, then output $z_{i,j}$.
2. Else if $\exists i, j \in T_k$ such that $\pi_{i,j}^k$ outputs (v_i, v_j), and for $s_m^{(m_1,m_2)} \triangleq \mathsf{RecInput}_m^{(m_1,m_2)}(v_m^{(m_1)}, v_m^{(m_2)})$, it holds that $\forall m \in \{i,j,k\}$, $s_m^{(i,k)} = s_m^{(j,k)} = s_m^{(i,j)} \neq \bot$, then (a) $\forall m \in \{i,j,k\}$, set $s''_m = s_m^{(i,j)}$, and (b) for $\ell \notin \{i,j,k\}$, set $s''_\ell = 0$, and (c) if $\exists$ distinct $m_1, m_2, m_3 \in \{i,j,k\}$ such that $s_\ell^{(m_1,m_2)} = \mathsf{RecInputNoBC}_\ell^{(m_1,m_2)}(c_\ell^{(m_3)}, v_\ell^{(m_1)}, v_\ell^{(m_2)}) \neq \bot$, then set $s''_\ell = s_\ell^{(m_1,m_2)}$, and (d) output $f(s''_1, \dots, s''_4)$.

Sketch of simulation and analysis. The main modification to the simulation proof from the previous subsection is that while simulating the above protocol, we use the modified extraction procedure described at the beginning of this subsection. The rest of the simulation is rather straightforward and follows the simulation procedure described in the previous subsection.

In the analysis, once again we design hybrid executions exactly as in previous proof. The main difference comes in the proof of indistinguishability of hybrids H_0 and H_1. We now look at a few cases. Suppose there is no majority among the commitment values distributed by the corrupt party, then by inspection of the protocol (particularly because of the use of $\mathsf{RecInputNoBC}$) it follows that the honest outputs are computed using extracted corrupt input 0. This is indeed the case in both hybrids H_0 and H_1. On the other hand, if there is a majority among these commitment values, and if there exists a matching decommitment (i.e., $\mathsf{code} = \mathsf{good}$), then we claim that the honest parties computed using the corrupt input that is consistent with the matching decommitment. To show this, first observe that there exists a PSM execution with honest clients from which every honest party obtains output. By inspection of the protocol (and particularly because of the use of RecViewNoBC), it follows that the output computed using the output of these PSM executions uses a corrupt input that is consistent with the matching decommitment. Now consider PSM executions in which one of the clients is malicious. Once again it follows from the use of RecInputNoBC along with an argument similar to the one in the previous proof, that in these cases as well, the final output is computed using a corrupt input that is consistent with the matching decommitment. Finally, when $\mathsf{code} = \mathsf{bad}$, we observe that outputs computed using the outputs of honest PSM executions (i.e., via Step 2) will use corrupt input equal to 0. Then following an argument similar to the one used in the previous proof, we have that the PSM executions which involve a corrupt client will either not be used to generate the final output, and will reconstruct an output that uses corrupt input 0.

The rest of the proof is quite straightforward and follows the same steps (with obvious modifications) as in the previous proof.

G More Details on 2-Round 4-Party Statistically Secure Protocol in the Preprocessing Model

G.1 2-Round Statistically Secure 4-Party Computation in the Preprocessing Model

In this section, we present a 2-round statistically secure computation protocol in the preprocessing model. We first present the simpler variant that uses a broadcast channel. See Appendix G for the final protocol. Recall T_i denotes the set $[4] \setminus \{i\}$, and $T_{i,j}$ denotes the set $[4] \setminus \{i, j\}$. We begin with an overview of the protocol.

Overview. The correlated randomness that we distribute to the parties is essentially a random pad per party, and a CNF share for each of these random pads. We stress that our correlated randomness essentially corresponds to a correct secret sharing of random pads, and in particular does not include MACs of the shares distributed. Somewhat surprisingly, such a "simple" correlated randomness is sufficient to yield a (relatively simple) protocol for 4-party secure computation in the preprocessing model. Below we describe the high level idea of our protocol.

In the online phase, each party simply broadcasts its input masked with the random pad it possesses. Then as before, parties run pairwise PSM protocols that essentially tries to reconstruct each party's input using the broadcasted messages and the CNF shares of each random pad held by the PSM clients. If everything is consistent, then the PSM evaluates the function on these reconstructed inputs. Then as in our 3-party secure-with-selective-abort protocol, we apply the "view reconstruction trick," i.e., we allow the PSM to try and reconstruct the correlated randomness that the PSM referee must possess. Then each party checks to find if there is a PSM execution that successfully evaluated the function (i.e., one that has a non-$\perp$ output), and if the reconstructed correlated randomness matches its correlated randomness. If such a PSM exists, then the party outputs the evaluation and terminates.

We now informally argue the security of our protocol. First, note that for every honest party, there exists a PSM execution (for e.g., one in which the two remaining honest parties act as PSM clients) that outputs a reconstructed correlated randomness that matches the honest party's correlated randomness. Further, it is easy to see that the evaluation performed by this PSM execution is correct. Given this, it remains to be shown that all other PSM executions produce outputs that either (a) will not be accepted by the honest party, or (b) will be consistent with the output of the above PSM. Suppose (1) the only way a malicious PSM client can force its PSM to produce a non-$\perp$ output is by supplying values consistent with the other (honest) client, and (2) the only way the output of a PSM execution in which one of the clients is malicious will be accepted is if the malicious client inputs correlated randomness as given to it. It is easy to see that security follows if both the properties hold. We enforce property (1) explicitly inside the PSM execution. Property (2) is enforced via use of the "view reconstruction trick."

We are now ready to formally describe the protocol for 4-party secure computation in the preprocessing model.

Correlated Randomness.

- For each $i \in [4]$, choose random r_i.
- For each r_i, let $\{r_{i,j}\}_{j \in T_i}$ denote the 1-private 3-party CNF sharing of r_i.
- For each $k \in [4]$, send to P_k values $(r_k, \mathsf{sh}_k = \{r_{i,j}\}_{i \in [4], j \in T_{k,i}})$.

Protocol. Let T_i denote the set $[4] \setminus \{i\}$, and let $T_{i,j}$ denote the set $[4] \setminus \{i, j\}$.

Round 1. Each P_i holding input x_i reconstructs $r_i = \bigoplus_{j \in T_i} r_{i,j}$, and broadcasts $y_i = x_i \oplus r_i$ to all parties. In addition, P_i also exchanges randomness with each P_j for a 2-client PSM protocol described below. For $i, k \in [4]$, let $w_k^{(i)}$ denote $(y_k^{(i)} = y_k, \mathsf{sh}_i = \{r_{k,\ell}^{(i)} = r_{k,\ell}\}_{k \in [4], \ell \in T_{i,k}})$.

Round 2. Each pair of parties (P_i, P_j) runs the following PSM protocol $\pi_{i,j}^{\ell}$ that delivers output to P_ℓ:

- Inputs: $w_i = \{w_k^{(i)}\}_{k \in [4]}$ from P_i, and $w_j = \{w_k^{(j)}\}_{k \in [4]}$ from P_j.
- If $\exists k \in [4]$ such that (1) $y_k^{(i)} \neq y_k^{(j)}$, or (2) $\exists t \notin \{i, j\}$ such that $r_{k,t}^{(i)} \neq r_{k,t}^{(j)}$, then output $\perp$ and terminate.
- For each $k \in [4], t \in T_k$, set $r'_{k,t}$ to be the unique non-$\perp$ value in $\{r_{k,t}^{(i)}, r_{k,t}^{(j)}\}$.
- For all $k \in [4]$, compute $r'_k = \bigoplus_{t \in T_k} r'_{k,t}$, and set $x'_k = y'_k \oplus r'_k$, where $y'_k = y_k^{(i)}$.
- Output $(z_{i,j}, \mathsf{sh}'_\ell)$, where $z_{i,j} = f(x'_1, \ldots, x'_4)$, and $\mathsf{sh}'_\ell = \{r'_{k,t}\}_{k \in [4], t \in T_{k,\ell}}$.

Output Computation. Each P_k finds some $i, j \in T_k$ such that $\pi_{i,j}^k$ outputs $(z_{i,j}, \mathsf{sh}_k)$ and then outputs $z_{i,j}$.

Fig. 4. 2-round 4-party protocol in the preprocessing model.

We prove the following lemma in Appendix G.2.

Lemma 3. *There exists a 2-round 4-party fully secure protocol (with guaranteed output delivery) for secure function evaluation in the preprocessing model that tolerates a single malicious party and uses broadcast in the first round only and whose correlated randomness complexity is $O(\ell)$ where ℓ is the length of each parties' input. The protocol provides statistical security for functionalities in* NC^1 *and computational security for general functionalities by making a black-box use of a pseudorandom generator.*

In Appendix G.3, we show how to remove the use of broadcast and prove the following theorem:

Theorem 7. *There exists a 2-round 4-party fully secure protocol (with guaranteed output delivery) for secure function evaluation over point-to-point channels in the preprocessing model that tolerates a single malicious party and whose correlated randomness complexity is $O(\ell)$ where ℓ is the length of each parties' input. The protocol provides statistical security for functionalities in* NC^1 *and computational security for general functionalities by making a black-box use of a pseudorandom generator.*

G.2 Proof of Lemma 3

We formally describe the simulation for corrupt party, say P_ℓ below.

Simulating corrupt P_ℓ. First, acting as the trusted party distributing correlated randomness, the simulator chooses uniformly random $r_{i,j}$ for each $i \in [4], j \in T_{\ell,i}$. Then it sets $r_\ell = \bigoplus_{j \in T_\ell} r_{\ell,j}$, and sends $(r_\ell, \mathsf{sh}_\ell = \{r_{i,j}\}_{i\in[4],j\in T_{\ell,i}})$ to P_ℓ. Next, for each $m \in T_\ell$, the simulator acting as P_m does the following:

- Pick y_m uniformly at random, and send y_m to P_ℓ over the broadcast channel.
- Send PSM randomness $r^{\mathrm{psm}}_{m,\ell,t}$ to P_ℓ over point-to-point channels for $t \in T_{m,\ell}$ if $m < \ell$.
- Receive y_ℓ over the broadcast channel from P_ℓ.
- Receive from P_ℓ PSM randomness $r^{\mathrm{psm}}_{\ell,m,t}$ for $t \in T_{m,\ell}$ if $m > \ell$.

The simulator extracts P_ℓ's input as $x_\ell = y_\ell \oplus r_\ell$, and sends x_ℓ to the trusted party, and receives back z_ℓ. Then for all $I \in T_\ell, j \in T_{\ell,i}$, $\mathcal{S}$ sets $z_{i,j} = (z_\ell, \mathsf{sh}_\ell)$.

In the next step, the simulator prepares to send the second round messages to P_ℓ by executing the following for all pairs (m_1, m_2) with $m_1 < m_2$.

Subroutine $\mathsf{PsmTrans}_\ell^{(m_1,m_2)}(z_{m_1,m_2})$

- Invoke PSM simulator $\mathcal{S}^{\mathrm{trans}}_{\pi^\ell_{m_1,m_2}}(1^\kappa, z_{m_1,m_2})$ to obtain transcript $\tau^{(m_1,m_2)}_{m_1}, \tau^{(m_1,m_2)}_{m_2}$.
- For all $m \in \{m_1, m_2\}$, acting as P_m sends $\tau^{(m_1,m_2)}_m$ to P_ℓ over point-to-point channels.

For each $i \in T_\ell, k \in T_{\ell,i}$, $\mathcal{S}$ receives PSM message $\tilde{\tau}^{(i,k)}_\ell$ from the adversary for execution $\pi^k_{\ell,i}$ delivering output to P_k. (Recall that $r^{\mathrm{psm}}_{\ell,i,k}$ denotes the PSM randomness used in execution $\pi^k_{\ell,i}$.) $\mathcal{S}$ then executes the following subroutine that extracts the value of r_ℓ implicitly used by P_ℓ in each PSM execution.

Subroutine $\mathsf{PsmExtract}_\ell(\{r^{\mathrm{psm}}_{\ell,i,k}, \tilde{\tau}_\ell^{(i,k)}\}_{i\in T_\ell, k\in T_{\ell,i}})$

- For each $i \in T_\ell, k \in T_{i,\ell}$, invoke PSM simulator $\mathcal{S}^{\mathrm{ext}}_{\pi^k_{\ell,i}}(1^\kappa, r^{\mathrm{psm}}_{\ell,i,k}, \tilde{\tau}_\ell^{(i,k)})$ to obtain output $\tilde{w}_{\ell,i,k} = \{\tilde{w}_m^{(\ell,i,k)}\}_{m\in[4]}$.
- If $\exists i \in T_\ell, k \in T_{\ell,i}$ such that $\tilde{w}_{\ell,i,k} = \bot$ (i.e., $\mathcal{S}^{\mathrm{ext}}_{\pi^k_{\ell,i}}$ failed), then output $\mathsf{psm\text{-}fail}$ and terminate.
- Parse $\tilde{w}_m^{(\ell,i,k)}$ as $(y_m^{(\ell,i,k)}, \mathsf{sh}_{\ell,i,k} = \{r^{(\ell,i,k)}_{m,t}\}_{m\in[4],t\in T_{m,\ell}})$.
- For each $i \in T_\ell, k \in T_{\ell,i}$, set $\mathsf{good}_{i,k} = 1$ if for all $m \in [4], t \in T_{m,\ell}$ it holds that $r^{(\ell,i,k)}_{m,t} = r_{m,t}$, else set $\mathsf{good}_{i,k} = 0$. Output $\{\mathsf{good}_{i,k}\}_{i\in T_\ell,k\in T_{\ell,i}}$.

If the output of $\mathsf{PsmExtract}$ is $\mathsf{psm\text{-}fail}$ then $\mathcal{S}$ outputs $\mathsf{psm\text{-}fail}$ and terminates. Else, $\mathcal{S}$ outputs whatever the adversary outputs, and terminates the simulation.

Analysis. First, we claim that the probability that $\mathcal{S}$ outputs $\mathsf{psm\text{-}fail}$ is negligible in κ. This follows directly from the security (more precisely, robustness property) of the PSM protocol π. In the following, we condition on the event that $\mathcal{S}$ did not output $\mathsf{psm\text{-}fail}$ in the simulated execution. Next, we claim that the corrupt party's output in the simulated execution is computed exactly as in the real execution. This follows from the fact that the extracted input of the adversary $x_\ell = y_\ell \oplus r_\ell$ equals the value x'_ℓ used by honest parties inside each PSM protocol that delivers output to P_ℓ. Given this, it follows from the security (more precisely, the privacy property) of the PSM protocol that the simulated PSM transcript is indistinguishable from the real transcript. Thus, we conclude that the view of the adversary in the real execution is indistinguishable from the view of adversary in the ideal execution.

Therefore, the simulated execution is indistinguishable from the real execution as long as the honest parties output identical values in the simulation and the real execution. We show that this is indeed the case. First note that for every honest P_k, there exists honest P_i, P_j such that P_k obtains a non-$\bot$ output from $\pi^k_{i,j}$. Furthermore, it is easy to verify that this output equals (z_k, sh_k), where sh_k is the set of shares of the random masks obtained from the trusted party that distributed correlated randomness, and z_k equals the output of the function computed on the extracted input of P_ℓ and the inputs of the honest parties. It remains to be shown that the output obtained from $\pi^k_{\ell,i}, \pi^k_{\ell,j}$ either equals (1) (z'_k, sh'_k) for $\mathsf{sh}'_k \neq \mathsf{sh}_k$, or (2) (z'_k, sh'_k) with $z'_k = z_\ell$ and $\mathsf{sh}'_k = \mathsf{sh}_k$, or (3) $\bot$. Observe that this is sufficient since in case (2) party P_k's output matches the output from $\pi^k_{i,j}$, while in cases (1) and (3) party P_k rejects this PSM output.

To prove the above we use the output of the subroutine $\mathsf{PsmExtract}$. Let us first analyze the execution $\pi^k_{\ell,i}$. (The analysis is identical for execution $\pi^k_{\ell,j}$.) Suppose $\mathsf{good}_{i,k} = 0$, then we claim that the output of $\pi^k_{\ell,i}$ will be $\bot$ or (z'_k, sh'_k) with $\mathsf{sh}'_k \neq \mathsf{sh}_k$. This is because the shares of the random masks held by honest P_i and P_k completely determine the actual shares of the random masks held by P_ℓ. In other words, when P_ℓ uses shares different from the ones distributed in the preprocessing stage, the value of P_k's shares reconstructed from shares of P_ℓ and P_i must differ from the shares held by P_k. It follows that P_k either obtains

$\perp$ as output from $\pi^k_{\ell,i}$ or simply rejects the output of $\pi^k_{\ell,i}$. Therefore, the claim holds. On the other hand, suppose it holds that $\mathsf{good}_{i,k} = 1$, then the shares of P_k recreated from P_ℓ's and P_i's shares exactly match the shares held by P_k. This, in particular, implies that the input of P_ℓ used in $\pi^k_{\ell,i}$ equals the value extracted by the simulator. Therefore, the output will be (z'_k, sh'_k) with $z'_k = z_k$ and $\mathsf{sh}'_k = \mathsf{sh}_k$. This completes the analysis of the simulation.

G.3 2-Round Statistically Secure 4-Party Computation in the Preprocessing Model Over Point-to-Point Channnels

Note: We get only statistical security is because the robust PSM can fail with negligible probability.
Recall T_i denotes the set $[4] \setminus \{i\}$, and $T_{i,j}$ denotes the set $[4] \setminus \{i, j\}$.

Overview. Observe that the protocol described in the previous section uses the broadcast channel only to distribute the y_ℓ values. A naïve attempt would simply be to replace this use of broadcast channel by letting the parties distribute these values over point-to-point channels. Unfortunately, the above variant of the protocol does not suffice (among other things) to guarantee output delivery to honest parties. Specifically, an adversary that sends different y_ℓ values can make every PSM execution deliver $\perp$ to the honest parties.

A natural next step is to replace the use of the broadcast channel by a protocol for broadcast (run concurrently with the PSM executions). It is possible to implement this idea because (1) parties make use of the broadcast channel only in the first round, and (2) there exists 2-round broadcast protocols tolerating a single corrupt party [25]. Unfortunately, this proposal also fails to achieve security.

We are now ready to formally describe the protocol for 4-party secure computation in the preprocessing model over point-to-point channels.

Correlated Randomness.
- For each $i \in [4]$, choose random r_i.
- For each r_i, let $\{r_{i,j}\}_{j \in T_i}$ denote the 1-private 3-party CNF sharing of r_i.
- For each $k \in [4]$, send to P_k values $(r_k, \mathsf{sh}_k = \{r_{i,j}\}_{i \in [4], j \in T_{k,i}})$.

Protocol. Let T_i denote the set $[4] \setminus \{i\}$, and let $T_{i,j}$ denote the set $[4] \setminus \{i, j\}$.

Round 1. Each P_i holding input x_i reconstructs $r_i = \bigoplus_{j \in T_i} r_{i,j}$.

$\star$ Each P_i sends $y_i = x_i \oplus r_i$ to every other P_j via point-to-point channels. Let P_j receive this value as $y_i^{(j)}$.

In addition, P_i also exchanges randomness with each P_j for a 2-client PSM protocol described below. For $i, k \in [4]$, let $w_k^{(i)}$ denote $(y_k^{(i)} = y_k, \mathsf{sh}_i = \{r_{k,\ell}^{(i)} = r_{k,\ell}\}_{k \in [4], \ell \in T_{i,k}})$.

Round 2. Each pair of parties (P_i, P_j) runs the following PSM protocol $\pi^\ell_{i,j}$ that delivers output to P_ℓ:

– Inputs: $w_i = \{w_k^{(i)}\}_{k\in[4]}$ from P_i, and $w_j = \{w_k^{(j)}\}_{k\in[4]}$ from P_j.
⋆ If $\exists k \in \{\ell, i, j\}$ such that $y_k^{(i)} \neq y_k^{(j)}$, or if $\exists k \in [4], t \notin \{i, j\}$ such that $r_{k,t}^{(i)} \neq r_{k,t}^{(j)}$, then output $\bot$ and terminate.
– For each $k \in [4], t \in T_k$, set $r'_{k,t}$ to be the unique non-$\bot$ value in $\{r_{k,t}^{(i)}, r_{k,t}^{(j)}\}$. Let $\mathsf{sh}'_\ell = \{r'_{k,t}\}_{k\in[4],t\in T_{k,\ell}}$.
⋆ For all $k \in [4]$, compute $r'_k = \bigoplus_{t\in T_k} r'_{k,t}$. For all $k \in \{\ell, i, j\}$, set $x'_k = y'_k \oplus r'_k$, where $y'_k = y_k^{(i)}$.
⋆ If for $k \notin \{\ell, i, j\}$ it holds that $y_k^{(i)} \neq y_k^{(j)}$, then output $\{\mathsf{sh}'_\ell, y'_i, y'_j, x'_i, x'_j, y_k^{(i)}, y_k^{(j)}, r'_k\}$ and terminate.
⋆ Output $(z_{i,j}, \mathsf{sh}'_\ell, y'_i, y'_j)$, where $z_{i,j} = f(x'_1, \ldots, x'_4)$.

⋆ **Output Computation.** Each P_k does the following:
– If $\exists i, j \in T_k$ such that $\pi_{i,j}^k$ outputs $(z_{i,j}, \mathsf{sh}_k, y_i^{(k)}, y_j^{(k)})$, then output $z_{i,j}$.
– Else if $\exists i, j, \ell \in T_k$ such that $\pi_{i,j}^k$ outputs $\{\mathsf{sh}_k, y_i^{(k)}, y_j^{(k)}, x'_i, x'_j, y_\ell^{(i)}, y_\ell^{(j)}, r'_\ell\}$ then compute $y'_\ell = \mathsf{majority}(y_\ell^{(i)}, y_\ell^{(j)}, y_\ell^{(k)})$, set $x'_\ell = y'_\ell \oplus r'_\ell$, and output $f(x'_1, \ldots, x'_4)$.

This completes the description of the protocol.

Intuition. The high level strategy used in the design of the protocol can be best explained as follows: Suppose party P_ℓ is corrupt.

– Let y'_ℓ denote the majority value among the y_ℓ values distributed by P_ℓ over point-to-point channels. (If no majority exists, then we simply set $y'_\ell = 0$.)
– Our protocol "extracts" the corrupt party's input as $x_\ell = y'_\ell \oplus r_\ell$, where r_ℓ is the random pad corresponding to P_ℓ obtained from the distributed correlated randomness.
– Then our protocol will force the final output of the honest parties to be computed using this extracted input for the corrupt party (and honest parties' inputs).

To show that this is indeed successfully implemented in our protocol, we will proceed by showing each of the following:

– Every PSM execution involving two honest clients delivering output to honest referee computes the output according to the above.
– There is no ambiguity in the output decision process due to PSM executions involving a corrupt client.
– The protocol is private.

For any i, j, k we say that PSM execution $\pi_{i,j}^k$ is either (1) awesome if its output is of the form $(z'_{i,j}, \mathsf{sh}'_k, y'_i, y'_j)$, or (2) good if its output is of the form $\{x'_i, x'_j, y_k^{(i)}, y_k^{(j)}, r'_k\}$, or (3) bad if its output is $\bot$. We now show that all three claims stated above hold.

Every PSM execution involving two honest clients delivering output to honest referee computes the output according to the above. Showing this is relatively straightforward. We split the analysis into two cases depending on whether the two honest clients agree on the y_ℓ value received from P_ℓ. Suppose they agree. Then it is easy to see that (1) the execution is awesome and (2) the output $z_{i,j}$ is computed using the majority y_ℓ value (which equals the value held by the two honest clients). On the other hand if the honest clients do not agree on the y_ℓ value, then in this case the execution will be good and once again, the final output computed by the honest party uses the corrupt party's input extracted using the majority of the y_ℓ values.

There is no ambiguity in the output decision process due to PSM executions involving a corrupt client. Obviously if the PSM execution involving corrupt client is bad, then it does not introduce any ambiguity in the output decision process. Our key observation is that if a PSM execution involving a corrupt party is not bad, then it must hold that the corrupt party provides a y_ℓ value that is consistent with other honest client's y_ℓ value. Next, note that this y_ℓ value is part of the output of the PSM execution, and further the honest referee discards the output of the PSM execution unless the y_ℓ value in the PSM output matches the y_ℓ value it holds. That is, the output of a non-bad PSM execution involving a corrupt client is used by the honest referee to compute its final output only if the honest client and the honest referee hold the same y_ℓ value, i.e., there is a well-defined majority value among the y_ℓ values. Furthermore, in this case, the PSM output is computed using the corrupt input that is extracted using the majority value. Thus, we conclude that the output of a PSM execution is used by the honest referee to compute its final output only if the PSM output is computed using a corrupt input that is extracted as the majority value.

The protocol is private. The key observation is that no PSM execution delivering output to a corrupt client is good. To see this, note that a PSM execution is good only if either (1) the party that is not involved in the PSM had sent different y values to clients, or (2) the clients supplied incorrect y values inside the PSM protocol. It then follows that when the PSM referee is corrupt neither (1) nor (2) can hold. Given that no PSM delivering output to corrupt referee is good, and the fact that a bad PSM does not reveal any information, it remains to only analyze the case when the PSM execution is awesome. Here, we only need to show that the output of PSM execution $\pi^\ell_{i,j}$, i.e., $z_{i,j}$, is always computed using the corrupt party's input extracted using the majority y_ℓ value. Indeed this is the case since the PSM execution is awesome only if the y_ℓ values held by the honest clients match, and therefore there is a well-defined majority y_ℓ value, and further from the protocol description it is evident that the corrupt input used in computing the PSM output is extracted using the majority y_ℓ value. Finally, we conclude by noting that the round 1 messages do not leak any information to the adversary since none of the PSMs leak any information about the random pads (that are distributed as part of the correlated randomness), and that the honest parties' round 1 broadcasts comprise of honest parties' inputs masked with these random pads and are hence hidden from the adversary.

We formalize the intuition described above by proving the following theorem.

Theorem 7. (restated) There exists a 2-round 4-party fully secure protocol (with guaranteed output delivery) for secure function evaluation over point-to-point channels in the preprocessing model that tolerates a single malicious party. The protocol provides statistical security for functionalities in NC^1 and computational security for general functionalities by making a black-box use of a pseudorandom generator.

Proof. We formally describe the simulation for corrupt party, say P_ℓ below.

Simulating corrupt P_ℓ. First, acting as the trusted party distributing correlated randomness, the simulator chooses uniformly random $r_{i,j}$ for each $i \in [4], j \in T_{\ell,i}$. Then it sets $r_\ell = \bigoplus_{j \in T_\ell} r_{\ell,j}$, and sends $(r_\ell, \mathsf{sh}_\ell = \{r_{i,j}\}_{i\in[4],j\in T_{\ell,i}})$ to P_ℓ. Next, for each $m \in T_\ell$, the simulator acting as P_m does the following:

- ⋆ Pick y_m uniformly at random, and send y_m to P_ℓ.
- – Send PSM randomness $r^{\mathrm{psm}}_{m,\ell,t}$ to P_ℓ over point-to-point channels for $t \in T_{m,\ell}$ if $m < \ell$.
- ⋆ Receive $y_\ell^{(m)}$ from P_ℓ.
- – Receive from P_ℓ PSM randomness $r^{\mathrm{psm}}_{\ell,m,t}$ for $t \in T_{m,\ell}$ if $m > \ell$.

⋆ To extract P_ℓ's input, $\mathcal{S}$ first computes $y'_\ell = \mathsf{majority}(y_\ell^{(i)}, y_\ell^{(i)}, y_\ell^{(i)})$ where $i,j,k \in T_\ell$ are distinct indices. Then it computes $x_\ell = y'_\ell \oplus r_\ell$, and sends x_ℓ to the trusted party, and receives back z_ℓ. For each $i \in T_\ell, j \in T_{\ell,i}$, $\mathcal{S}$ sets $z_{i,j} = (z_\ell, \mathsf{sh}_\ell, y_i, y_j)$ if $y'_\ell = y_\ell^{(i)} = y_\ell^{(j)}$, else it sets $z_{i,j} = \bot$.

In the next step, the simulator prepares to send the second round messages to P_ℓ by executing the following for all pairs (m_1, m_2) with $m_1 < m_2$.

<u>SUBROUTINE $\mathsf{PsmTrans}_\ell^{(m_1,m_2)}(z_{m_1,m_2})$</u>

- Invoke PSM simulator $\mathcal{S}^{\mathrm{trans}}_{\pi^\ell_{m_1,m_2}}(1^\kappa, z_{m_1,m_2})$ to obtain transcript $\tau^{(m_1,m_2)}_{m_1}, \tau^{(m_1,m_2)}_{m_2}$.
- For all $m \in \{m_1, m_2\}$, acting as P_m sends $\tau^{(m_1,m_2)}_m$ to P_ℓ over point-to-point channels.

For each $i \in T_\ell, k \in T_{\ell,i}$, $\mathcal{S}$ receives PSM message $\tilde{\tau}_\ell^{(i,k)}$ from the adversary for execution $\pi^k_{\ell,i}$ delivering output to P_k. (Recall that $r^{\mathrm{psm}}_{\ell,i,k}$ denotes the PSM randomness used in execution $\pi^k_{\ell,i}$.) $\mathcal{S}$ then executes the following subroutine that extracts the value of r_ℓ implicitly used by P_ℓ in each PSM execution.

<u>SUBROUTINE $\mathsf{PsmExtract}_\ell(\{r^{\mathrm{psm}}_{\ell,i,k}, \tilde{\tau}_\ell^{(i,k)}\}_{i\in T_\ell, k\in T_{\ell,i}})$</u>

- For each $i \in T_\ell, k \in T_{i,\ell}$, invoke PSM simulator $\mathcal{S}^{\mathrm{ext}}_{\pi^k_{\ell,i}}(1^\kappa, r^{\mathrm{psm}}_{\ell,i,k}, \tilde{\tau}_\ell^{(i,k)})$ to obtain output $\tilde{w}_{\ell,i,k} = \{\tilde{w}_m^{(\ell,i,k)}\}_{m\in[4]}$.
- If $\exists i \in T_\ell, k \in T_{\ell,i}$ such that $\tilde{w}_{\ell,i,k} = \bot$ (i.e., $\mathcal{S}^{\mathrm{ext}}_{\pi^k_{\ell,i}}$ failed), then output $\mathsf{psm\text{-}fail}$ and terminate.
- Parse $\tilde{w}_m^{(\ell,i,k)}$ as $(y_m^{(\ell,i,k)}, \mathsf{sh}_{\ell,i,k} = \{r_{m,t}^{(\ell,i,k)}\}_{m\in[4],t\in T_{m,\ell}})$.

- For each $i \in T_\ell, k \in T_{\ell,i}$, set $\mathsf{good}_{i,k} = 1$ if for all $m \in [4], t \in T_{m,\ell}$ it holds that $r^{(\ell,i,k)}_{m,t} = r_{m,t}$, else set $\mathsf{good}_{i,k} = 0$. Output $\{\mathsf{good}_{i,k}\}_{i \in T_\ell, k \in T_{\ell,i}}$.

If the output of PsmExtract is psm-fail then $\mathcal{S}$ outputs psm-fail and terminates. Else, $\mathcal{S}$ outputs whatever the adversary outputs, and terminates the simulation.

Analysis. First, we claim that the probability that $\mathcal{S}$ outputs psm-fail is negligible in κ. This follows directly from the security (more precisely, robustness property) of the PSM protocol π. In the following, we condition on the event that $\mathcal{S}$ did not output psm-fail in the simulated execution. Next, we claim that the corrupt party's output in each simulated PSM execution is computed exactly as in the real execution. Consider the real PSM execution $\pi^\ell_{i,j}$. It is easy to see that if P_i and P_j hold different values for y_ℓ, then PSM execution $\pi^\ell_{i,j}$ delivers $\perp$ as output to P_ℓ. On the other hand, when P_i and P_j hold identical values for y_ℓ, then this value equals the majority value, and hence the extracted value in the simulation is the one used inside $\pi^\ell_{i,j}$ to compute output. Given the above, it follows from the security (more precisely, the privacy property) of the PSM protocol that the simulated PSM transcript is indistinguishable from the real transcript. Thus, we conclude that the view of the adversary in the real execution is indistinguishable from the view of adversary in the ideal execution.

Therefore, the simulated execution is indistinguishable from the real execution as long as the honest parties output identical values in the simulation and the real execution. We show that this is indeed the case. For any i, j, k we say that PSM execution $\pi^k_{i,j}$ is either (1) awesome if its output is of the form $(z'_{i,j}, \mathsf{sh}'_k, y'_i, y'_j)$, or (2) good if its output is of the form $\{x'_i, x'_j, y^{(i)}_k, y^{(j)}_k, r'_k\}$, or (3) bad if its output is $\perp$. First note that for every honest P_k, there exists honest P_i, P_j such that $\pi^k_{i,j}$ is either awesome or good. We now split the analysis into two cases depending on the output of $\pi^k_{i,j}$.

- Suppose $\pi^k_{i,j}$ is awesome. Since for honest P_i, P_j it holds that $y'_i = y^{(k)}_i$ and $y'_j = y^{(k)}_j$ and $\mathsf{sh}'_k = \mathsf{sh}_k$, party P_k outputs $z_{i,j}$ unless there is another awesome execution say $\pi^k_{\ell,i}$ whose output is $z'_{i,j} \neq z_{i,j}$. Suppose such $\pi^k_{\ell,i}$ exists. First, note that sh_k and sh_j together completely determine the value of the random masks distributed in the preprocessing phase. Further, P_ℓ has to input $\{y'_m\}_{m \in [4]}$ that is consistent with P_j's view, otherwise the execution $\pi^k_{\ell,i}$ would not be awesome. In other words, both the random masks and values $\{y'_m\}_{m \in [4]}$ used inside $\pi^k_{\ell,i}$ are identical to the ones used inside $\pi^k_{i,j}$. Since these values completely determine the output, it must hold that $z'_{i,j} = z_{i,j}$. It remains to be shown that $z_{i,j}$ equals the output computed in the simulated execution as well. To prove this, first we note that since $\pi^k_{i,j}$ is awesome, it must hold that $y^{(i)}_\ell = y^{(j)}_\ell = y'_\ell$. In particular, this means that $y'_\ell = \mathsf{majority}(y^{(i)}_\ell, y^{(j)}_\ell, y^{(k)}_\ell)$, and $x'_\ell = r_\ell \oplus y'_\ell$. It is easy to see that the input value extracted by the simulator is identical to x'_ℓ and therefore, P_k's output in the real and simulated executions are identically distributed.

– Suppose $\pi^k_{i,j}$ is good. Let the output of $\pi^k_{i,j}$ be $\{\mathsf{sh}'_k, y'_i, y'_j, x'_i, x'_j, y^{(i)}_\ell, y^{(j)}_\ell, r'_\ell\}$ and let $y'_\ell = \mathsf{majority}(y^{(i)}_\ell, y^{(j)}_\ell, y^{(k)}_\ell)$, $x'_\ell = y'_\ell \oplus r'_\ell$, $x'_k = x_k$, and $z'_k = f(x'_1, \ldots, x'_4)$. First note that $\mathsf{sh}'_k = \mathsf{sh}_k$, $y'_i = y^{(k)}_i$, and $y'_j = y^{(k)}_j$ all hold. Next, observe that z'_k is identical to the output of P_k in the simulated execution. Thus, it is sufficient to prove that output of PSM executions $\pi_{\ell,i}, \pi_{\ell,j}$ is either discarded or results in final output z'_k. In the following we analyze the PSM execution $\pi^k_{\ell,i}$. (The analysis of $\pi^k_{\ell,j}$ is identical.) There are three cases to handle depending on whether $\pi^k_{\ell,i}$ is $\mathsf{awesome}$, good, or bad. It is easy to see that when $\pi^k_{\ell,i}$ is bad, the output is discarded. Next, it is easy to see if $\pi^k_{\ell,i}$ is $\mathsf{awesome}$, then the output of $\pi^k_{\ell,i}$ is discarded unless it is of the form $(z_{\ell,i}, \mathsf{sh}_k, y^{(k)}_\ell, y_i)$. Thus, it is sufficient to prove that $z_{\ell,i} = z'_k$. Indeed this is case since the P_ℓ input $y^{(k)}_\ell$ as its input value, and further since $\pi^k_{\ell,i}$ is $\mathsf{awesome}$, it must hold that $y^{(i)}_\ell = y^{(k)}_\ell$, i.e., $\mathsf{majority}(y^{(k)}_\ell, y^{(i)}_\ell, y^{(j)}_\ell) = y^{(i)}_\ell$. Since the value $y^{(i)}_\ell$ is used for computing P_ℓ's input inside $\pi^k_{\ell,i}$ it follows that the output $z_{\ell,i}$ must necessarily equal z'_k. Finally we analyze the case when $\pi^k_{\ell,i}$ is good. It is easy to see that the output of $\pi^k_{\ell,i}$ is discarded unless it equals $(\mathsf{sh}_k, y^{(k)}_\ell, y^{(k)}_i, x'_\ell, x'_j, y^{(i)}_j, y^{(\ell)}_j, r'_j)$. Since $\pi^k_{\ell,i}$ is good, it must also hold that $y^{(k)}_\ell = y^{(i)}_\ell$. This in turn implies that value x'_ℓ must equal the majority value extracted in the simulation execution. Next, note that $y_j = y'_j = \mathsf{majority}(y^{(\ell)}_j, y^{(i)}_j, y^{(k)}_j)$ holds since $y^{(i)}_j = y^{(k)}_j$. This in turn combined with the fact $\pi^k_{\ell,i}$ output $\mathsf{sh}'_k = \mathsf{sh}_k$ implies that $x_j = x'_j = r'_j \oplus y'_j$ also holds. Thus we conclude that the output $f(x'_1, \ldots, x'_4)$ must equal z'_k since identical inputs were used to evaluate f in both cases. This completes the analysis of the simulation.