# 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 $\mathrm{NC}^1$. 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 $\mathrm{NC}^1$.

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, 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, if we wish to obtain secure computation protocols with perfect/statistical security, 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 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. In the full version, we show 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).

**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) and (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) and (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 (VSS) and private simultaneous message (PSM) protocols. We also describe the secret sharing schemes we use.

**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 $\sigma$ 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 $\sigma$, it holds that $s_i = s$.*

**Commitment** *Except with probability negligible in $\sigma$, 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 $r = r^{\text{psm}}$, 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 $\pi$ for a function $f : \{0,1\}^{\ell \times m} \to \{0,1\}^*$ is defined by $R(\ell)$, a randomness length parameter, $m$ message algorithms $A_1, \ldots, A_m$ and a reconstruction algorithm $\mathsf{Rec}$, such that the following requirements hold.

- *Correctness:* for every input length $\ell$, 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 $\ell$, 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 $\ell$ also as security parameter, as in [17]; this is without loss of generality, by padding inputs to the required length.)

A *robust* PSM protocol $\pi$ 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 $\mathsf{Rec}$ on inputs $A_T(x_T^*, r), A_{\overline{T}}(x_{\overline{T}}, r)$ is statistically close to the "real-world" output of $\mathsf{Rec}$ 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 $\mathsf{Rec}$ 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 $\mathsf{Rec}$ 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. In this paper $n \in \{3, 4\}$. We denote by $T_i$ (resp. $T_{i,j}$) 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 corrupted 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 $r^{\text{psm}}_{i,j} = r^{\text{psm}}_{j,i}$ 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* $\mathrm{NC}^1$ *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.

*Simulation sketch.* Denote the corrupt party by $P_\ell$. 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 $z_\ell$.

Next the simulator invokes the PSM simulator $\mathcal{S}^{\mathrm{trans}}_{\pi_{i,j}}$ (guaranteed by the privacy property) on inputs $z_\ell$ 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 $P_\ell$ 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 $P_\ell$ 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 $P_\ell$ 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 $P_\ell$ 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 $P_\ell$, 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 $P_\ell$ 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 $P_\ell$ 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 $P_\ell$ in the PSM execution $\pi_{\ell,j}$. It follows by simple inspection that the criterion used to add $i$ to $S_\ell$ 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. $\square$

**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 $r^{\text{psm}}_{i,j}$. (For instance, by letting $P_i$ pick $r^{\text{psm}}_{i,j}$ and send $r^{\text{psm}}_{i,j}$ 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 $\sigma$ garbled circuits where $\sigma$ 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. 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 $\sigma$ 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 the full version, 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. For a detailed step-by-step overview of the protocol, please see the full version where we prove:

**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 $\pi$ for general secure computation. Let us further set up some notation related to protocol $\pi$. Let $A_{i,j}^{(r)}$ denote the algorithm specified by protocol $\pi$ 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. $\omega_i$) 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 established. 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 $\omega_1$ 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 $\pi$ 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 $\omega_1$, 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 $\omega_1$ 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 $\mathsf{fail}_1$ and terminate.

- Compute $\tilde{m}^{(1)}_{1,3} \leftarrow A^{(1)}_{1,3}(1, \perp, \tilde{\omega})$.
- For $j = 2, 3, 4$, send message $\tilde{m}^{(1)}_{1,j}$ to $P_j$ in round 1.
- Receive round 1 messages $m^{(1)}_{2,1}$, $m^{(1)}_{3,1}$, $m^{(1)}_{4,1}$, from other parties. Do not send any round 2 messages to any party. Receive round 2 messages $m^{(2)}_{2,1}$, $m^{(2)}_{3,1}$, $m^{(2)}_{4,1}$, from other parties and terminate the protocol.
- Compute and output $y'_0 \leftarrow A^{(3)}_{1,1}(0, \{\{m^{(1)}_{k,i}\}_{k\in T_1}, \{m^{(2)}_{k,i}\}_{k\in\{2,4\}}\}; \omega_1)$, $y'_1 \leftarrow A^{(3)}_{1,1}(1, \{\{m^{(1)}_{k,i}\}_{k\in T_1}, \{m^{(2)}_{k,i}\}_{k\in\{3,4\}}\}; \tilde{\omega})$.

First, we claim that corrupt $P_1$ does not output $\mathsf{fail}_1$ 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 $\pi$ against an (all-powerful) $P_4$. Clearly, if there exists no $\tilde{\omega}$ such that the output of $A^{(1)}_{1,4}$ 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 $\omega$) 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 $A^{(3)}_{1,1}$ 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 $\pi$ 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 $A^{(3)}_{1,1}$ 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 $\pi$ 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). □

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. For a formal description of the protocol, and how to remove the use of broadcast, please see the full version where we prove:

**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.