From Fairness to Full Security in Multiparty Computation*

Ran Cohen† Iftach Haitner‡ Eran Omri§ Lior Rotem¶

December 4, 2021

Abstract

In the setting of secure multiparty computation (MPC), a set of mutually distrusting parties wish to jointly compute a function, while guaranteeing the privacy of their inputs and the correctness of the output. An MPC protocol is called *fully secure* if no adversary can prevent the honest parties from obtaining their outputs. A protocol is called *fair* if an adversary can prematurely abort the computation, however, only before learning any new information.

We present efficient transformations from fair computations to fully secure computations, assuming a constant fraction of honest parties (e.g., 1% of the parties are honest). Compared to previous transformations that require linear invocations (in the number of parties) of the fair computation, our transformations require super-logarithmic, and sometimes even super-constant, such invocations. The main idea is to delegate the computation to random committees that invoke the fair computation. Apart from the benefit of uplifting security, the reduction in the number of parties is also useful, since only committee members are required to work, whereas the remaining parties simply "listen" to the computation over a broadcast channel.

One application of these transformations is a new δ-bias coin-flipping protocol, whose round complexity has a super-logarithmic dependency on the number of parties, improving over the linear-dependency protocol of Beimel, Omri, and Orlov (Crypto 2010). A second application is a new fully secure protocol for computing the Boolean OR function, with a super-constant round complexity, improving over the protocol of Gordon and Katz (TCC 2009) whose round complexity is linear in the number of parties.

Finally, we show that our positive results are in a sense optimal, by proving that for some functionalities, a super-constant number of (sequential) invocations of the fair computation is necessary for computing the functionality in a fully secure manner.

Keywords: multiparty computation; fairness; identifiable abort, security reductions.

*A preliminary version of this work appeared at *SCN 2018* [25].

†Reichman University. E-mail: cohenran@idc.ac.il. Research supported in part by NSF grant no. 2055568. Most of this work was done while the author was a post-doctoral researcher at Tel Aviv University, supported by ERC starting grant 638121.

‡The Blavatnik School of Computer Science, Tel Aviv University. E-mail: iftachh@tauex.tau.ac.il. Member of the Check Point Institute for Information Security. Research supported by ERC starting grant 638121 and the Israel Science Foundation grant 666/19.

§Department of Computer Science, Ariel University. Ariel Cyber Innovation Center (ACIC). E-mail: omrier@ariel.ac.il. Research supported by ISF grant 152/17, and by the Ariel Cyber Innovation Center in conjunction with the Israel National Cyber directorate in the Prime Minister's Office.

¶School of Computer Science and Engineering, Hebrew University of Jerusalem. E-mail: lior.rotem@cs.huji.ac.il. Supported by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities, the European Union's Horizon 2020 Framework Program (H2020) via an ERC Grant (Grant No. 714253), and the Israel Science Foundation (Grant No. 483/13).

Contents

1 Introduction

In the setting of secure multiparty computation (MPC), a set of mutually distrusting parties wish to jointly compute a function of their inputs, while guaranteeing the privacy of their local inputs and the correctness of the output. The security definition of such a computation has numerous variants. A major difference between the variants, which is the focus of this work, is the ability of an adversary to prevent the honest parties from completing the computation by corrupting a subset of the parties. According to the *full-security* variant, an adversary cannot prevent the honest parties from receiving their output.[1] A more relaxed security definition called *fairness*, allows an adversary to prematurely abort the computation, but only *before* it has learned any information from the computation. Finally, *security with abort* allows an adversary to prevent the honest parties from receiving the output, even *after* it has learned the output, but never to learn anything more.[2]

A common paradigm for constructing a protocol that provides a high security guarantee (e.g., full security) for a given functionality f, is to start with constructing a protocol for f of a low security guarantee (e.g., security with abort), and then to "uplift" the security of the protocol via different generic transformations (e.g., the GMW compiler from semi-honest security to malicious security). Hence, finding such security-uplifting transformations is an important research question in the study of MPC. In this work, we study such security-uplifting transformations from security with abort and fairness to full security.

It is known that when the *majority* of the parties are honest, security with abort can be uplifted to fairness. Given an n-party functionality f, let $\mathsf{SS}_{\mathsf{out}}(f)$ denote the functionality that outputs secret shares of $y = f(x_1, \ldots, x_n)$ using an $\lceil n/2 \rceil$-out-of-n error-correcting secret-sharing scheme (ECSS).[3] Assume that $\mathsf{SS}_{\mathsf{out}}(f)$ can be computed securely with abort. In case the adversary aborts the computation of $\mathsf{SS}_{\mathsf{out}}(f)$, it does not learn any new information, since it can only obtain less than $n/2$ shares. Whereas in case the adversary does not abort, it cannot prevent the honest parties from reconstructing the *correct* output, thus completing the computation. Similarly, assume $\mathsf{SS}_{\mathsf{out}}(f)$ can be securely computed with *identifiable abort*,[4] then the security of computing $\mathsf{SS}_{\mathsf{out}}(f)$ can be uplifted to a fully secure computation of f via the following player-elimination technique: All parties iteratively compute $\mathsf{SS}_{\mathsf{out}}(f)$ with identifiable abort, such that in each iteration either all honest parties obtain the output, or the adversary aborts the computation at the cost of revealing the identity of a corrupted party. After at most $t + 1$ iterations, it is guaranteed that the computation will successfully complete. Security with identifiable abort can be reduced to security with abort, assuming one-way functions, via a generic reduction [33]. More efficient generic reductions in terms of round complexity appear in [57, 45], using stronger hardness assumptions.

In case no honest majority is assumed, it is impossible to *generically* transform security with (identifiable) abort to full security, and even not to fairness; every functionality can be computed with abort [33] (assuming oblivious transfer exists), but some functionalities cannot be fairly computed [21, 54]. In contrast, fairness can be uplifted to full security also in the no-honest-majority

[1] This property is also referred to as *guaranteed output delivery*.

[2] Throughout the paper, unless explicitly stated otherwise, by security with abort we mean *unanimous* abort where all honest parties reach agreement on whether to abort or not. We note that since we consider a broadcast model, the weaker notion of *non-unanimous* abort [34, 26] in which some honest parties may abort while other receive their output, can be uplifted to unanimous abort in a single broadcast round.

[3] A $(t + 1)$-out-of-n secret-sharing scheme is error correcting, if the reconstruction algorithm outputs the correct secret even when up to t shares are arbitrarily modified. ECSS schemes are also known as *robust secret sharing*.

[4] Same as security with abort, except that upon a premature abort, all honest parties identify a corrupted party.

case [22] (assuming one-way functions exist),[5] by first uplifting the security to fairness with identifiable abort, and then invoking (up to) $t+1$ fair computations of f with identifiable abort.

In the setting of large-scale computation, the linear dependency on number of corruptions forms a bottleneck, and might blow-up the round complexity of the fully secure protocol. In this work, we explore how, and to what extent, this linear dependency can be reduced.

1.1 Our Results

Our main positive result is highly efficient reductions from full security to fair computation, assuming that the fraction of honest parties is constant (e.g., 1% of the parties are honest). We show how to compute in a fully secure manner an n-party functionality f^n, by fairly computing a related n'-party functionality $f^{n'}$ for $\omega(1)$ sequential times, where $n' = \omega(\log(\kappa))$ (e.g., $n' = \log^*(\kappa)\cdot\log(\kappa)$) and κ is the security parameter. For some functionalities, we only need to be able to compute the functionality $f^{n'}$ in a security-with-abort manner (no fairness is needed). Throughout, we assume the static-corruption model, where the corrupted parties are determined *before* the protocol begins.

Apart from the obvious benefit of being security-uplifting (from fairness to full security), the reduction in the number of parties is also useful, i.e., only $n' = \omega(\log(\kappa))$ parties are required to work in the protocol, whereas the remaining parties simply "listen" to the computation over a broadcast channel. The efficiency of secure protocols is typically proportional to the number of parties (in some cases, e.g., [6, 16], the dependency is exponential). Furthermore, for implementations that are only δ-close to being fair (i.e., the real-world computation is δ-distinguishable from the ideal-world computation, denoted δ-fair below), the error parameter δ is typically a function of the number of parties. Hence, even given a fully secure implementation (or δ-close to being fully secure, denoted δ-fully-secure below) of a functionality, applying the above reductions can improve both the security error and the efficiency (see the applications part below for concrete examples). The reductions presented in this paper are depicted in Figure 1, alongside previously known reductions.

To keep the following introductory discussion simple, we focus below on no-input, public-output functionalities (a single output is given to all parties). A less detailed description of the reductions to security with abort and the reductions for the case of with-input functionalities can be found in Section 1.3. We start by describing the reduction from a fully secure computation of a no-input functionality (e.g., coin flipping) to a fair computation of this functionality, and an application of this reduction to fair coin flipping. We then describe a lower bound on the number of rounds in which such a reduction (from fully secure) invokes the fair functionality.

Our protocols make use of Feige's lightest-bin protocol for committee election [31] (see Section 1.2). For integers $n' < n$ and for $0 < \beta < \beta' < 1$, Feige's protocol is used by n parties, β fraction of which are corrupted, to elect a committee of size n', whose fraction of corrupted parties is at most β'. We denote by $\mathsf{err}(n, n', \beta, \beta') = \frac{n}{n'} \cdot e^{-\frac{(\beta'-\beta)^2 n'}{2(1-\beta)}}$ the error probability of Feige's protocol. Note that for $n' = \omega(\log(\kappa))$ it holds that $\mathsf{err}(n, n', \beta, \beta')$ is negligible (in κ), and for $n' = \Omega(\log(\kappa))$ it holds that $\mathsf{err}(n, n', \beta, \beta')$ is inverse-polynomial (the latter is applicable to the δ-bias coin-flipping application in which δ is inverse-polynomial).

Our results in the no-honest-majority setting hold under the assumption that *enhanced trapdoor permutations (TDP)* and *collision-resistant hash functions (CRH)* exist. Given a no-input

[5] Unless stated otherwise, we assume that parties can communicate over a broadcast channel. If a broadcast channel is not available, identifiable abort cannot be achieved generically [22], and indeed, some functionalities can be fairly computed, but not with full security [22, 24].

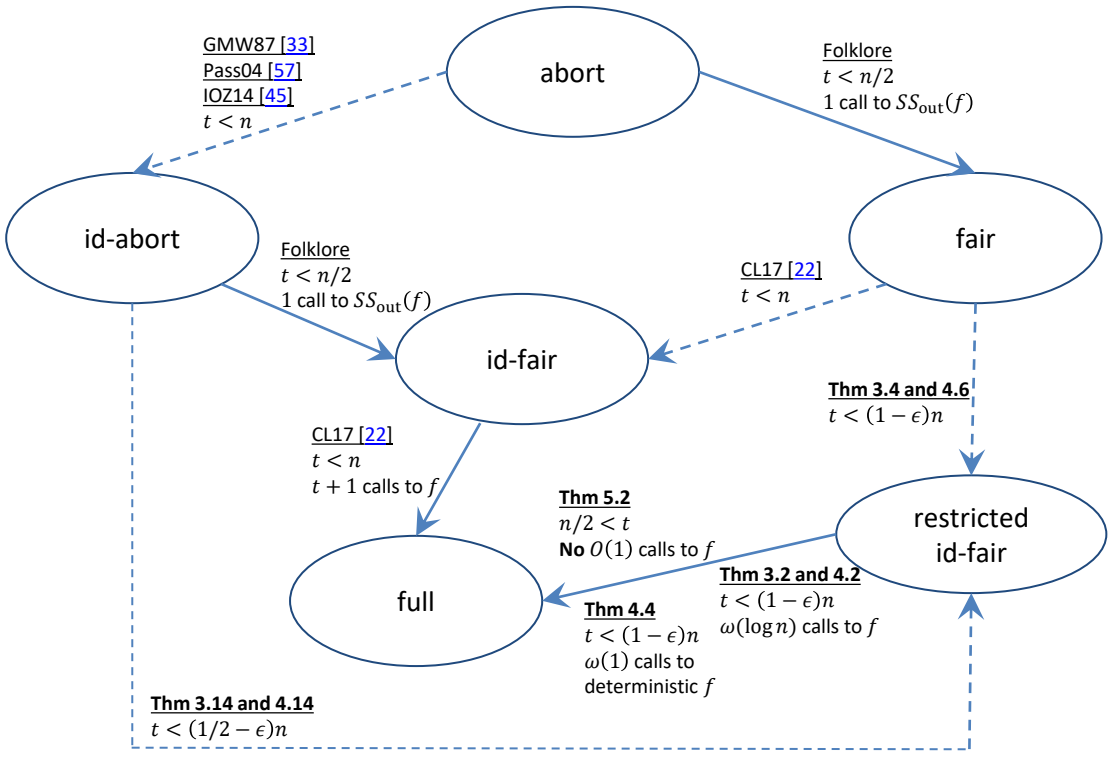

Figure 1: Reductions between security notions. Solid arrows refer to black-box reductions with respect to the functionality (i.e., a hybrid model) whereas dashed arrows refer to non-black-box reductions (i.e., a protocol compiler). *Restricted id-fair* refers to fairness where the set of parties who can abort the computation is restricted to a designated subset.

functionality f, let f^n denote its n-party variant: the output contains n copies of the common output.

Theorem 1.1 (fairness to full security, no-input case, informal)**.** *Let f be a no-input functionality, let $n' < n \in \mathbb{N}$, let $0 < \beta < \beta' < 1$, let $t = \beta n$, let $t' = \beta' n'$, and let $\mathsf{err} = \mathsf{err}(n, n', \beta, \beta')$. If $f^{n'}$ can be δ'-fairly computed by an r'-round protocol π' tolerating t' corruptions, then the following hold.*

1. *Assuming TDP and CRH, f^n can be computed with $(t' \cdot \delta' + \mathsf{err})$-full-security, tolerating t corruptions by an $O(t' \cdot r')$-round protocol. Furthermore, if π' is δ'-fully-secure, then the resulting protocol is $(\delta' + \mathsf{err})$-fully-secure and has $O(t' + r')$ rounds.*

2. *For $\beta' < 1/2$ and $n' = \varphi(\kappa) \cdot \log(\kappa)$ for $\varphi = \Omega(1)$, if π' can be computed ℓ-times in parallel, for $\ell = \kappa^c$ (for some universal constant c), then f^n can be computed with $(\varphi(\kappa)^2 \cdot \ell \cdot \delta' + \mathsf{err})$-full-security, unconditionally, tolerating t corruptions by an $O(\varphi(\kappa)^2 \cdot r')$-round protocol. Furthermore, the computation is black-box in the protocol π'.*[6]

[6] Following [41], by a black-box access to a protocol we mean a black-box usage of a semi-honest MPC protocol computing its next-message function.

The idea underlying the above reduction is quite simple. To achieve a fully secure computation of an n-party functionality f^n, we first choose a small committee of size n', using an information-theoretically secure committee-election protocol. The computation is then delegated to this small committee, which in turn, securely computes the functionality with fairness and identifiable abort. Since, the computation of the small committee might abort, we might need to repeat this process several times, while eliminating the aborting parties. See Section 1.2 for more details.

Application to coin flipping. As an application of the above type of reduction, we show how to improve on the round complexity of δ-bias coin-flipping protocols. The n-party, no-input, public-output, coin-flipping functionality f^n_{cf} outputs to all parties a uniformly distributed bit $b \in \{0,1\}$. A δ-bias, t-secure, n-party coin-flipping protocol is a real-world, polynomial-time, n-party protocol that emulates the ideal functionality f^n_{cf} up to a δ distinguishing distance, even in the face of up to t corruptions.

Cleve [21] has given a lower bound that relates the bias in any r-round coin-flipping protocol to $1/r$. Awerbuch et al. [3] constructed an r-round, t-secure, $O(t/\sqrt{r})$-bias coin-flipping protocol for an arbitrary number of parties n and $t < n$. This was improved by Beimel et al. [6], who gave an r-round, t-secure, $O(1/\sqrt{r-t})$-bias coin-flipping protocol for the case that $t = \beta n$ for some constant $0 < \beta < 1$. Recently, Beimel et al. [7] showed that for a "large" number of parties, $n = r^{\varepsilon}$ (for a constant $\varepsilon > 0$), any r-round protocol can be efficiently biased by $\tilde{\Omega}(1/\sqrt{r})$. We remark that r-round coin-flipping protocols of bias $o(t/\sqrt{r})$ are known when the number of parties is "small," $n < \log\log r$, or when the difference between corrupted and honest parties is constant [55, 6, 38, 1, 16]. None of these protocols, however, deals with a large number of parties when a $\beta > 0.51$ fraction of them are malicious. For this case, it is not known how to obtain a bias that is independent of the number of corruptions. Using Theorem 1.1, we are able to improve upon [6] by replacing the linear dependency on t with a super-logarithmic dependency on the security parameter κ.

Corollary 1.2 (informal)**.** *Assume that TDP and CRH exist. Let $n' < n$ be integers, and let $0 < \beta < \beta' < 1$ be constants. If there exists an n'-party, δ'-bias, r'-round coin-flipping protocol π' tolerating $t' = \beta' n'$ corrupted parties, then there exists an n-party, $(\delta' + \mathsf{err}(n, n', \beta, \beta'))$-bias, $O(t' + r')$-round coin-flipping protocol, tolerating $t = \beta n$ corrupted parties.*

Concretely, by using the protocol of Beimel et al. [6], we obtain that for every $\varphi(\kappa) = \omega(1)$, every sufficiently large n (greater than $\varphi(\kappa) \cdot \log(\kappa)$), every $0 < \beta < 1$, and every efficiently computable $r : \mathbb{N} \mapsto \mathbb{N}$ there exists an n-party, $r(\kappa)$-round, $O(1/\sqrt{r(\kappa) - \varphi(\kappa) \cdot \log(\kappa)})$-bias, βn-secure coin-flipping protocol.

Lower bound on the number of sequential fair calls. We prove that some functionalities, and in particular coin flipping, achieving full-security requires a super-constant number of *functionality rounds*, i.e., rounds in which a fair ideal functionality is invoked, even if a constant fraction of parties are honest. Namely, the (super-)logarithmic multiplicative overhead in the round complexity, induced by Theorem 1.1 (Item 1) for achieving negligible (or inverse-polynomial) error, cannot be reduced to constant.

The lower bound is proven in a hybrid model in which an ideal computation with fairness and identifiable abort of the functionality is carried out by a trusted party. For a no-input functionality f^n, the model allows different subsets of parties (committees) to invoke the trusted party *in parallel* (in the same functionality round), such that only committee members can abort the call

to the trusted party that is made by the committee. We assume that the outputs of such parallel invocations, which consist of bit-values and/or identities of the aborting parties, are given at the *same time* to all n parties, unless an invocation is made by an all-corrupted committee, which can first see the output of the other parallel invocations *before* deciding upon its action.

The above model is more optimistic than the one we can actually prove to exist, assuming a fair protocol for computing the functionality at hand (hence, proving lower bounds is harder in this model). Actually, the no-honest-majority part of Theorem 1.1 (Item 1) can be pushed further in this model to match the lower bound given below. See Section 5.1 for further discussion regarding this model.

Theorem 1.3 (necessity of super-constant sequential fair calls, informal)**.** *The following holds in the hybrid model in which any subset of the parties can invoke the trusted party that* fairly *computes the coin-flipping functionality. Let π be a coin-flipping protocol in this model that calls the trusted party in a constant number of rounds (i.e., in each round, the trusted party can be invoked many times in parallel by different subsets). Then, for any $1/2 < \beta < 1$, there exists an efficient fail-stop adversary controlling βn parties that noticeably biases the output of the protocol.*

Note that in this model, fully secure coin-flipping protocols do exist (e.g., as we show in Theorem 1.1, by invoking the trusted party in a super-logarithmic number of rounds).

1.2 Our Techniques

We start with describing the techniques underlying our positive results, focusing on the no-input case for the sake of clarity of the presentation. Later below, we discuss the ideas underlying the lower bound on round complexity.

Upper bound. Let f^n be some n-party (no-input, public-output) functionality, and let π be an n-party, r-round protocol that computes f^n with fairness, tolerating $t < n$ corruptions. It was shown by [22] that π can be compiled into a protocol that computes f^n with fairness and identifiable abort. The original compilation uses the technique of [33] and is inefficient in terms of round complexity. However, using the constant-round, bounded-concurrent, zero-knowledge techniques of Pass [57] (that require TDP and CRH), the resulting protocol has $O(r)$ rounds. Having this compilation in mind, we henceforth consider the goal of uplifting fairness with identifiable abort to full security. Let π be a protocol that computes f^n with fairness and identifiable abort tolerating $t = \beta \cdot n$ corruptions. A naïve way for achieving full security is using the above mentioned player-elimination technique to obtain a fully secure computation of f^n. This, however, comes at a cost in terms of round complexity. Specifically, the resulting protocol will run in $O(t \cdot r)$ rounds.

In the following, we explain how the security-uplifting transformation can be kept efficient in terms of round complexity. Our transformation builds on the player-elimination technique and works given the following three components: (i) a method to select a small subset (committee) $\mathcal{C}$ of n' parties that contains at most $t' = \beta' \cdot |\mathcal{C}|$ corrupted parties (for arbitrary small $\beta' > \beta$), (ii) an n'-party, r'-round protocol π' that computes $f^{n'}$ with fairness and identifiable abort, and (iii) a monitoring procedure for all n parties to verify the correctness of an execution of π' run by the committee members. In such a case, we could get a simple security-uplifting reduction with a low round complexity (assuming $r' \leq r$). Specifically, in order to compute f^n with full security, we would select a committee $\mathcal{C}$, let the parties in $\mathcal{C}$ execute π' with full security using the player-elimination technique, while the remaining parties monitor the execution and receive

the final output from the committee members. Since player elimination will only be applied to committee members, it may be applied at most t' times. Hence, the resulting protocol will run in $O(t' \cdot r')$ rounds. Below, we explain how to select a committee $\mathcal{C}$, and how the execution of the protocol π' can be monitored by non-committee parties. Whether an appropriate protocol π' exists depends on the functionality at hand.

Our key tool for electing the committee is Feige's lightest-bin protocol [31]. This is a single-round protocol, secure against computationally unbounded adversaries, ensuring the following. If n parties with up to $\beta \cdot n$ corruptions use the protocol to elect a committee $\mathcal{C}$ of size n', then for all $\beta' > \beta$, the fraction of corrupted parties in the committee is at most β', with all but probability $\mathsf{err}(n, n', \beta, \beta') = \frac{n}{n'} \cdot e^{-\frac{(\beta'-\beta)^2 n'}{2(1-\beta)}}$. In particular, for $n' = \omega(\log(\kappa))$ Feige's protocol succeeds with all but negligible probability (in κ). The beauty of this protocol is in its simplicity, as parties are simply instructed to select a random bin (out of n/n' possible ones), and the elected committee are the parties that chose the lightest bin.

We now turn to explain how the non-committee parties can monitor the work of the committee members. In the no-input setting that we have discussed so far, things are quite simple. Recall that all our protocols assume a broadcast channel, which allows the non-committee parties to see all communication among committee members.[7] Now, all that is needed is that when the protocol terminates, the non-committee parties can verify that they obtain the correct output from the computation. To this end, we start the protocol with committee members being publicly committed to a random string (used as their randomness in the execution). Then, as the protocol ends, a committee member notifies all parties of the output it received by proving in zero knowledge that it has followed the prescribed protocol using the randomness it committed to.

Proving security of the above reduction raises a subtle technical issue. Whenever a computation by the committee is invoked, it is required that all parties will obtain the output (either a genuine output or an identity of a corrupted committee member); however, only corrupted committee members are allowed to abort the computation. This property is not captured by the standard definition of fairness with identifiable abort, where every corrupted party can abort the computation. We therefore introduce a new ideal model with fairness and restricted identifiable abort that models this property. In this ideal model, the trusted party is parametrized by a subset $\mathcal{C} \subseteq [n]$. The adversary, controlling parties in $\mathcal{I} \subseteq [n]$, can abort the computation only if $\mathcal{I} \cap \mathcal{C} \neq \emptyset$, by revealing the identity of a corrupted party $i^* \in \mathcal{I} \cap \mathcal{C}$. This means that if $\mathcal{I} \cap \mathcal{C} = \emptyset$ this ideal model provides full security; however, in case $\mathcal{C} \subseteq \mathcal{I}$, no security is provided, and the adversary gets to choose the output.[8]

The proof consists of two steps. Initially, full security is reduced to fairness with restricted identifiable abort. This is done by electing a super-logarithmic committee $\mathcal{C}$ using Feige's protocol, and iteratively invoking the trusted party for computing f^n with fairness with restricted identifiable abort, parametrized by $\mathcal{C}$, until the honest parties obtain the output. Next, fairness with restricted identifiable abort is reduced to fairness. This is done by compiling (in a similar way to the GMW compiler) the protocol π' for computing $f^{n'}$ with fairness into a protocol π for computing f^n with fairness with restricted identifiable abort.

Lower bound. Recall that our lower bound is given in the hybrid model in which a trusted party computes the coin-flipping functionality with fairness and restricted identifiable abort (as presented before, Theorem 1.3). In this model, in addition to standard communication rounds, a

[7] Private messages should be encrypted before being sent over the broadcast channel.

[8] In the with-input setting Section 1.3, the adversary also obtains the input values of all honest parties.

protocol also has *functionality rounds* in which different committees (subsets) of the parties invoke the trusted party.

Consider an n-party coin-flipping protocol π in this hybrid model with a constant number of functionality rounds. The heart of the proof is showing that if none of the committees is large, i.e., has more than $\log(\kappa)$ parties, then the protocol can be biased noticeably. The proof is completed by showing that since π has only a constant number of functionality rounds, an adversary can force all calls made by large committees to abort, and thus, attacking arbitrary protocols reduces to the no-large-committee case.

To prove the no-large-committees case, we transform the n-party coin-flipping protocol π in the hybrid model, into a two-party coin-flipping protocol ψ in the standard model. By Cleve [21], there exists an attack on protocol ψ. Hence, we complete the proof by showing how to transform the attack on ψ (guaranteed by Cleve [21]) into an attack on π. The aforementioned protocol transformation goes as follows: partition the n parties of π into two subsets, $\mathcal{S}_0$ of size βn and $\mathcal{S}_1 = [n] \setminus \mathcal{S}_0$. The two-party protocol $\psi = (\mathsf{P}_0, \mathsf{P}_1)$ emulates a random execution of π by letting party P_0 emulate the parties in $\mathcal{S}_0$ and party P_1 emulate the parties in $\mathcal{S}_1$. The calls to the trusted party are emulated by P_0 as follows: let $\mathcal{C}_1, \ldots, \mathcal{C}_\ell$ be the (small) committees that invoke the trusted party, in parallel, in a functionality round. In protocol ψ, party P_0 sends ℓ uniformly distributed bits, each bit in a different round, and the parties interpret these bits as the output produced by the coin-flipping functionality. At the end of the protocol, each party outputs the output of the first party of π in its control. If P_0 aborts while emulating a functionality round, i.e., when it is supposed to send the output bit of a committee $\mathcal{C}$, party P_1 continues as if the first party in $\mathcal{C}$ (for simplicity, we assume this party is in $\mathcal{S}_0$) aborts the call to the trusted party in π, and the rest of the parties in $\mathcal{S}_0$ abort immediately after the call to the trusted party. If P_0 aborts in a round that emulates a communication round in π, party P_1 continues the emulation of π as if all parties in $\mathcal{S}_0$ abort. Party P_0 handles an abort by P_1 analogously.

By Cleve [21], there exists a round i^* such that one of the parties in ψ can bias the protocol merely by deciding, depending on its view, whether to abort in round i^* or not.[9] Assume, without loss of generality, that the attacking party is P_0, and the round i^* is a functionality round (other cases translate directly to attacks on π). The core difference between the ability of an adversary corrupting party P_0 in ψ from that of an adversary corrupting the parties in $\mathcal{S}_0$ in π, is that the adversary in ψ can decide whether to abort *before* sending the i^*'th message. This raises a subtle issue, since the i^*'th message corresponds to an output of the coin-flipping functionality in π, in response to a call made by some committee $\mathcal{C}$. Yet, if the adversary in π controls *all* parties in $\mathcal{C}$, he can abort *after* seeing the output of the call to trusted party made by $\mathcal{C}$ and the results of all other parallel calls, while still preventing other parties from getting the output of the call made by $\mathcal{C}$. We conclude the proof by showing that if the corrupted subset $\mathcal{S}_0$ is chosen at random, then it contains all parties in the relevant committee with a noticeable probability, and thus, the attack on π goes through.

1.3 Additional Results

In the above discussion, we only reviewed our reductions from full security to fairness for the no-input case. This was done for the sake of clarity; however, in this paper we also deal with arbitrary

[9] The attacker of [21] either aborts at round i^* or at round $i^* + 1$, but the transformation to the above attacker is simple (see Section 5.3).

functionalities (with input). We remark that the lower bound for the no-input case, described above, applies also to the with-input case.

1.3.1 Full Security to Fairness: Arbitrary Functionalities (with Inputs)

The case of functionalities with inputs is somewhat more involved than that of no-input functionalities. As in the no-input case, our fully secure computation of an n-party functionality f^n is done by delegating the computation to a small committee that computes a related n'-party functionality with fairness. However, when considering functionalities with inputs, parties outside the committee cannot reveal their inputs to committee members, but still need to make sure that the right input was used in the computation performed by committee members. This can be done using secret-sharing schemes and commitments. Note that non-committee parties take a bigger role in the computation now. However, corrupted parties outside the committee should never be able to cause the protocol to prematurely terminate, as otherwise the number of rounds would depend on the number of corruptions among all parties and not only committee members. The above becomes even more challenging when wishing to have a few committees perform the computation in parallel. Here, it must also be verified that each party provides *the same* input to all committees.

Considering the no-honest-majority case, we let each party P_i secret share its input x_i in an n'-out-of-n' secret sharing, publicly commit to every share, and send each decommitment value, encrypted, to the corresponding committee member. We define $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f^n)$ to be the n'-party functionality, parametrized by a vector of commitments $(c_i^1,\ldots,c_i^{n'})$ for every P_i, where c_i^j is a commitment to the j'th share of x_i. The functionality receives as input the decommitments of each c_i^j, reconstructs the decommitted values to obtain the n-tuple $(x_1,\ldots,x_n)$, computes $y = f^n(x_1,\ldots,x_n)$, and outputs y in the clear (see Figure 3).

By having the parties publicly commit to shares of their inputs (using a perfectly binding commitment) and send the decommitment values to the committee members, corrupted committee members cannot change the values corresponding to honest parties (otherwise the decommit will fail and the cheating committee member will be identified). Preventing corrupted parties from sending invalid decommitments to honest committee members is external to the functionality and must be part of the protocol. In addition to TDP and CRH, we assume non-interactive perfectly binding commitment schemes exist.[10] We prove the following.

Theorem 1.4 (fairness to full security, informal)**.** *Let f^n be an n-party functionality, let $n' = \varphi(\kappa)\cdot\log(\kappa)$ for $\varphi = \Omega(1)$, let $0 < \beta < \beta' < 1$, let $t = \beta n$, let $t' = \beta' n'$, and let $\mathsf{err} = \mathsf{err}(n, n', \beta, \beta')$. The following hold assuming TDP, CRH, and non-interactive perfectly binding commitment schemes.*

1. *If $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f^n)$ can be δ'-fairly computed by an r'-round protocol, tolerating t' corruptions, then f^n can be computed with $(t'\cdot\delta'+\mathsf{err})$-full-security, tolerating t corruptions, by an $O(t'\cdot r')$-round protocol.*

2. *If $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f^n)$ can be δ'-fairly computed by an r'-round protocol, tolerating $n'-1$ corruptions, ℓ-times in parallel, for $\ell = \kappa^c$ (for some universal constant c), then f^n can be computed with $(\varphi(\kappa)^2\cdot\ell\cdot\delta'+\mathsf{err})$-full-security, tolerating t corruptions, by an $O(\varphi(\kappa)^2\cdot r')$-round protocol.*

[10] Although non-interactive perfectly binding commitments can be constructed from one-way permutations, in our setting, one-way functions are sufficient. This follows since Naor's commitments [56] can be made non-interactive in the common random string (CRS) model, and even given a weak CRS (a high min-entropy common string). A high min-entropy string can be constructed by n parties, without assuming an honest majority, using the protocol from [37] that requires $\log^*(n)+O(1)$ rounds.

In the honest-majority setting, a similar result can be achieved with the transformation only requiring black-box access to the fair protocol, and the resulting security being unconditional. Furthermore, the transformation becomes much simpler with an honest majority and relies solely on ECSS scheme. We denote by $\mathsf{SS}_{\mathsf{in\text{-}out}}^{n\to(t',n')}(f^n)$, for $t' < n'/2$, the n'-party functionality that receives secret shares of an n-tuple $(x_1,\ldots,x_n)$, reconstructs the inputs, computes $y = f(x_1,\ldots,x_n)$ and outputs secret shares of y (see Figure 4). See Section 4.2 for more details.

Reducing a logarithmic factor. When considering functionalities with inputs, it is possible to use generic techniques (see, for example, [39, Sec. 2.5]) and assume without loss of generality that the functionality is deterministic and has a public output (i.e., all parties receive the same output). In this case, we show how to reduce an additional logarithmic factor from the number of fair computations performed by the committee, compared to the no-input case. The parties start by electing a random, (super-)logarithmic committee $\mathcal{C}$, of size $m = \varphi(\kappa)\cdot\log(\kappa)$, for some $\varphi(\kappa) \in \Omega(1)$ (e.g., $\varphi(\kappa) = \log^*(\kappa)$). However, instead of sharing the inputs with the committee members, the protocol considers all sufficiently large sub-committees, i.e., all subsets of $\mathcal{C}$ of size $n' = m - \log(\kappa)/\varphi(\kappa)$. Next, every party secret shares its input to each of the sub-committees, and each of the sub-committees computes, in parallel, the functionality $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f)$ with fairness and identifiable abort. It is important for each party to prove in zero knowledge that the same input value is shared across all sub-committees, in order to ensure the same output value in all computations. We show that in this case: (1) there are polynomially many sub-committees, (2) with overwhelming probability, no sub-committee is fully corrupted, and (3) if the adversary aborts the fair computations in all sub-committees, then $\log(\kappa)/\varphi(\kappa)$ corrupted parties must be identified. It follows that after $\varphi^2(\kappa)$ iterations the protocol is guaranteed to successfully terminate.

In order to prove security of this construction, we generalize the notion of *fairness with restricted identifiable abort* to the with-input setting. The ideal model is parametrized by a list of subsets $\mathcal{C}_1,\ldots,\mathcal{C}_\ell \subseteq [n]$, such that if one of the subsets is fully corrupted, i.e., $\mathcal{C}_i \subseteq \mathcal{I}$ for some $i \in [\ell]$ (where $\mathcal{I}$ is the set of corrupted parties), then no security is provided (the adversary gets all inputs and determines the output). If one of the subsets is fully honest, i.e., $\mathcal{C}_i \cap \mathcal{I} = \emptyset$ for some $i \in [\ell]$, then the adversary cannot abort the computation. Otherwise, the adversary is allowed to abort the computation by revealing a corrupted party in each subset; however, only before it has learned any new information. See Section 2.5 for more details.

Application to fully secure multiparty Boolean OR. An application of the above reductions is a fully secure protocol for n-party Boolean OR. Gordon and Katz [35] constructed a fully secure protocol, tolerating $t < n$ corruptions, that requires $O(t)$ rounds. Using Theorem 1.4, we show how to achieve (any) super-constant round complexity when the fraction of corruptions is constant.

Corollary 1.5 (informal)**.** *Under the assumptions in Theorem 1.4, the n-party Boolean OR functionality can be computed with full security tolerating $t = \beta n$ corruptions, for $0 < \beta < 1$, with round complexity $O(\log^*(\kappa))$.*

Application to a best-of-both-worlds type result. Another application is to a variant of the protocol of Ishai et al. [43] that guarantees t-full-security assuming an honest majority and *t-full-privacy* otherwise.[11] Their idea is to repeatedly compute $\mathsf{SS}_{\mathsf{out}}(f^n)$, using a secure protocol

[11] t-full-privacy means that the adversary does not learn any additional information other than what it can learn from $t+1$ invocations of the ideal functionality, with fixed inputs for the honest parties.

with identifiable abort, and use the player-elimination approach until the honest parties obtain the secret shares and reconstruct the result. It follows that the round complexity in [43] is $O(t)$. The above reduction suggests an improvement both to the round complexity of the protocol and to the privacy it guarantees.

Corollary 1.6 (informal). *Let f^n be an n-party functionality and let $t = \beta n$ for $0 < \beta < 1$, and consider the assumptions as in Theorem 1.4. Then, there exists a single protocol π, with round complexity $O(\log^*(\kappa) \cdot \log(\kappa))$, such that:*

1. *π computes f^n with $O(\log^*(\kappa) \cdot \log(\kappa))$-full-privacy.*
2. *If $\beta < 1/2$, then π computes f^n with full security.*

Application to uplifting partially identifiable abort to full security. Finally, we improve a recent transformation of Ishai et al. [46, Thm. 3 and 4] from partially identifiable abort[12] to full security in the honest-majority setting. In [46], the computation of $\mathsf{SS}_{\mathsf{in\text{-}out}}^{n\to(t',n')}(f^n)$ with partially identifiable abort is carried out iteratively by a committee, initially consisting of all the parties, until the output is obtained. In case of abort, all the identified parties (both honest and corrupted) are removed from the committee. It follows that the number of iterations in [46] is $O(n)$.

Corollary 1.7 (informal). *Let f^n be an n-party functionality, let $n' = \log^*(\kappa) \cdot \log(\kappa)$, let $0 < \beta < \beta' < 1/2$, let $t = \beta n$ and $t' = \beta' n'$, and let π' be an r-round protocol that securely computes $\mathsf{SS}_{\mathsf{in\text{-}out}}^{n\to(t',n')}(f^n)$ with β'-partially identifiable abort, tolerating t' corruptions. Then, f^n can be computed with full security, tolerating t corruptions, by a $O(t' \cdot r)$-round protocol that uses the protocol π' in a black-box way.*

1.4 Additional Related Work

The MPC literature contains many examples of reductions from strong security notions to weaker ones, e.g., [33, 18, 41, 42], to name but a few. Recently, Ishai et al. [46] have presented a formal framework for studying (black-box) transformations between different security notions (see further examples therein). All of our results in the honest-majority setting can be stated in the framework of [46].

Security with identifiable abort was first explicitly used by Aumann and Lindell [2]; however, it was widely used in the literature implicitly, especially in the realm of fairness [43, 40, 61, 35, 6, 36, 5, 38, 22, 45, 4, 1, 16]. Kiayias et al. [48] have recently defined security with *publicly identifiable abort*, where a subset of the parties perform a computation, and all parties (even outside of the subset) identify a corrupted party in case of abort. This definition is different from the definition we introduce of *restricted identifiable abort*, since unlike our definition, in [48] non-working parties do not provide input and do not receive output; such parties can only identify a corrupted party in case of abort.

The idea of electing a small committee to perform a computation was initially used in [14], and has been considered in numerous settings, such as: leakage-resilient secure computation [9, 10], large-scale MPC [11, 12, 30, 13], leader election [51, 47], Byzantine agreement [49, 50, 52, 15], and distributed key-generation [19].

[12] A computation has α-partially identifiable abort [46], if in case the adversary aborts the computation, a subset of parties is identified, such that at least an α-fraction of the subset is corrupted.

Lindell and Rabin [53] have recently considered a slightly different model, in which a secure protocol must have a fixed and a priori known *committal round*, where the effective inputs of the corrupted parties are determined. They showed that fair protocols cannot be constructed in this model without an honest majority. We note that our constructions of fair protocols over committed inputs do not contradict this impossibility result, since we consider the standard model, where the simulator may choose not to use the values that are committed by the corrupted parties in the protocol, and may decide on the effective inputs based on the behavior of the real-world adversary at *any* round.

1.5 Open Questions

In the no-input, no-honest-majority setting, there is a logarithmic gap between the $\omega(\log(\kappa))$ number of sequential fair calls required by our reductions, and our $\omega(1)$ lower bound on this number. The source for this gap is in the difference between the pessimistic model we use in our reductions, and the more optimistic (yet, as far as we know, possible) model we use in our lower bound (see more details in Section 5.1). Finding the right security model to capture the power of such protocols, and finding tight reductions in this setting, is an interesting open question.

Paper Organization

Basic definitions can be found in Section 2. Our reductions from full security to fairness for no-input functionalities are given in Section 3, and for functionalities with inputs in Section 4. The lower bound on the number of sequential fair calls is given in Section 5.

2 Preliminaries

2.1 Notations

We use calligraphic letters to denote sets, uppercase for random variables, lowercase for values, boldface for vectors, and sans-serif (e.g., A) for algorithms (i.e., Turing Machines). For $n \in \mathbb{N}$, let $[n] = \{1, \cdots, n\}$. Let poly denote the set all positive polynomials and let PPT denote a probabilistic algorithm that runs in *strictly* polynomial time. A function $\nu \colon \mathbb{N} \mapsto [0,1]$ is *negligible*, denoted $\nu(\kappa) = \mathrm{neg}(\kappa)$, if $\nu(\kappa) < 1/p(\kappa)$ for every $p \in \mathsf{poly}$ and large enough κ. The statistical distance between two random variables X and Y over a finite set $\mathcal{U}$, denoted $\mathrm{SD}(X,Y)$, is defined as $\frac{1}{2} \cdot \sum_{u \in \mathcal{U}} |\Pr[X = u] - \Pr[Y = u]|$. Given a random variable X, we write $x \leftarrow X$ to indicate that x is selected according to X.

Two distribution ensembles $X = \{X(a,\kappa)\}_{a \in \{0,1\}^*, \kappa \in \mathbb{N}}$ and $Y = \{Y(a,\kappa)\}_{a \in \{0,1\}^*, \kappa \in \mathbb{N}}$ are computationally δ-indistinguishable (denoted $X \equiv_c^\delta Y$) if for every non-uniform polynomial-time distinguisher A there exists a function $\nu(\kappa) = \mathrm{neg}(\kappa)$, such that for every $a \in \{0,1\}^*$ and $\kappa \in \mathbb{N}$

$$|\Pr[\mathsf{A}(X(a,\kappa), 1^\kappa) = 1] - \Pr[\mathsf{A}(Y(a,\kappa), 1^\kappa) = 1]| \leq \delta(\kappa) + \nu(\kappa).$$

In case δ is negligible, we say that X and Y are computationally indistinguishable and denote $X \equiv_c Y$. The distribution ensembles X and Y are (statistically) δ-close (denoted $X \equiv_s^\delta Y$) if $\mathrm{SD}(X,Y) \leq \delta + \nu(\kappa)$ for a negligible function ν, and statistically close (denoted $X \equiv_s Y$) is they are δ-close and δ is negligible. X and Y are perfectly δ-close (denoted $X \equiv^\delta Y$) if $\mathrm{SD}(X,Y) \leq \delta$. In case $\delta = 0$, i.e., if X and Y are identically distributed, denote $X \equiv Y$.

We denote by n the number of participating parties in a protocol, by t an upper bound on the number of corrupted parties, and by κ the security parameter. As standard in the context of large-scale secure computation (see, e.g., [28, 29]), we assume that n and κ are polynomially related, i.e., $n = \kappa^c$ for some constant $c > 0$ (possibly $c < 1$). In particular, this means that a functionality f is in fact an ensemble of functionalities $\{f_n\}_{n\in\mathbb{N}}$, where f_n is an n-party functionality; for example, in the Boolean OR functionality, for every $n \in \mathbb{N}$, $f_n(x_1, \ldots, x_n) = x_1 \vee \ldots \vee x_n$. We refer the reader to [13] for a discussion on functionalities and protocols in the large-scale setting.

2.2 Secret Sharing

A (threshold) secret-sharing scheme [59] is a method in which a dealer distributes shares of some secret to n parties such that t colluding parties do not learn anything about the secret, and any subset of $t + 1$ parties can fully reconstruct the secret.

Definition 2.1 (secret sharing)**.** *A $(t+1)$-out-of-n secret-sharing scheme over a message space $\mathcal{M}$ consists of a pair of algorithms* $(\mathsf{Share}, \mathsf{Recon})$ *satisfying the following properties:*

1. ***t-privacy:*** *For every $m \in \mathcal{M}$, and every subset $\mathcal{I} \subseteq [n]$ of size $|\mathcal{I}| \leq t$, the distribution of $\{s_i\}_{i\in\mathcal{I}}$ is independent of m, where $(s_1, \ldots, s_n) \leftarrow \mathsf{Share}(m)$.*
2. ***$(t+1)$-reconstructability:*** *For every $m \in \mathcal{M}$, every subset $\mathcal{I} \subseteq [n]$ of size $t + 1$, every $\boldsymbol{s} = (s_1, \ldots, s_n)$ and every $\boldsymbol{s}' = (s'_1, \ldots, s'_n)$ such that $\Pr_{\boldsymbol{S}\leftarrow\mathsf{Share}(m)}[\boldsymbol{S} = \boldsymbol{s}] > 0$, $\boldsymbol{s}_{\mathcal{I}} = \boldsymbol{s}'_{\mathcal{I}}$, and $\boldsymbol{s}'_{\bar{\mathcal{I}}} = \perp^{|\bar{\mathcal{I}}|}$, it holds that $m = \mathsf{Recon}(\boldsymbol{s}')$.*

An error-correcting secret-sharing (ECSS) scheme is a secret-sharing schemes, in which the reconstruction is guaranteed to succeed even if up to t shares are faulty. This primitive has also been referred to as *robust secret sharing* or as *honest-dealer VSS* [58, 20, 27].

Definition 2.2 (error-correcting secret sharing)**.** *A $(t+1)$-out-of-n error-correcting secret-sharing scheme (ECSS) over a message space $\mathcal{M}$ consists of a pair of algorithms* $(\mathsf{Share}, \mathsf{Recon})$ *satisfying the following properties:*

1. ***t-privacy:*** *As in Definition 2.1.*
2. ***Reconstruction from up to t erroneous shares:*** *For every $m \in \mathcal{M}$, every $\boldsymbol{s} = (s_1, \ldots, s_n)$, and every $\boldsymbol{s}' = (s'_1, \ldots, s'_n)$ such that $\Pr_{\boldsymbol{S}\leftarrow\mathsf{Share}(m)}[\boldsymbol{S} = \boldsymbol{s}] > 0$ and $|\{i \mid s_i = s'_i\}| \geq n - t$, it holds that $m = \mathsf{Recon}(\boldsymbol{s}')$.*

ECSS can be constructed with perfect correctness when $t < n/3$ using Reed-Solomon decoding [8] and with a negligible error probability when $t < n/2$ by authenticating the shares using one-time MAC [58]. In case $t \geq n/2$ it is impossible to construct a $(t + 1)$-out-of-n ECSS scheme, or even a secret-sharing scheme that identifies cheaters [44].

2.3 Security Definitions

We provide the basic definitions for secure multiparty computation according to the real/ideal paradigm, for further details see [32]. Informally, a protocol is considered secure if whatever an adversary can do in the real execution of protocol, can be done also in an ideal computation, in which an uncorrupted trusted party assists the computation.

Definition 2.3 (functionalities)**.** *An n-party* functionality *is a random process that maps vectors of n inputs to vectors of n outputs.*[13] *Given an n-party functionality $f\colon (\{0,1\}^*)^n \mapsto (\{0,1\}^*)^n$, let $f_i(\boldsymbol{x})$ denote its i'th output coordinate, i.e., $f_i(\boldsymbol{x}) = f(\boldsymbol{x})_i$. A functionality f has* public output*, if the output values of all parties are the same, i.e., for every $\boldsymbol{x} \in (\{0,1\}^*)^n$, $f_1(\boldsymbol{x}) = f_2(\boldsymbol{x}) = \ldots = f_n(\boldsymbol{x})$, otherwise f has* private output*.*

A no-input n-party functionality $f\colon (\{0,1\}^*)^n \mapsto (\{0,1\}^*)^n$ is a functionality in which the input of every party is the empty string λ. That is, f is computed as $(y_1,\ldots,y_n) \leftarrow f(\lambda,\ldots,\lambda;r)$ over random coins r. In case f is a no-input functionality with public output, it can be defined for any number of parties; we denote by f^n the functionality f when defined for n parties.

2.3.1 Execution in the Real World

An n-party protocol $\pi = (\mathsf{P}_1,\ldots,\mathsf{P}_n)$ is an n-tuple of probabilistic polynomial-time interactive Turing machines. The term *party* P_i refers to the i'th interactive Turing machine. Each party P_i starts with input $x_i \in \{0,1\}^*$ and random coins $r_i \in \{0,1\}^*$. Without loss of generality, the input length of each party is assumed to be the security parameter κ. An *adversary* A is another interactive TM describing the behavior of the corrupted parties. It starts the execution with input that contains the identities of the corrupted parties and their private inputs, and possibly an additional auxiliary input. The parties execute the protocol in a synchronous network. That is, the execution proceeds in rounds: each round consists of a *send phase* (where parties send their messages from this round) followed by a *receive phase* (where they receive messages from other parties). The adversary is assumed to be *rushing*, which means that it can see the messages the honest parties send in a round before determining the messages that the corrupted parties send in that round.

The parties can communicate in every round over a broadcast channel or using a fully connected point-to-point network. We consider two models for the communication lines between the parties: In the *authenticated-channels* model (used in the computational setting), the communication lines are assumed to be ideally authenticated but not private (and thus the adversary cannot modify messages sent between two honest parties but can read them). In the *secure-channels* model (used in the information-theoretic setting), the communication lines are assumed to be ideally private (and thus the adversary cannot read or modify messages sent between two honest parties).

Throughout the execution of the protocol, all the honest parties follow the instructions of the prescribed protocol, whereas the corrupted parties receive their instructions from the adversary. The adversary is considered to be *malicious*, meaning that it can instruct the corrupted parties to deviate from the protocol in any arbitrary way. At the conclusion of the execution, the honest parties output their prescribed output from the protocol, the corrupted parties output nothing and the adversary outputs an (arbitrary) function of its view of the computation (containing the views of the corrupted parties). The view of a party in a given execution of the protocol consists of its input, its random coins, and the messages it sees throughout this execution.

Definition 2.4 (real-model execution)**.** *Let $\pi = (\mathsf{P}_1,\ldots,\mathsf{P}_n)$ be an n-party protocol and let $\mathcal{I} \subseteq [n]$ denote the set of indices of the parties corrupted by A. The* joint execution of π under $(\mathsf{A},\mathcal{I})$ in the real model*, on input vector $\boldsymbol{x} = (x_1,\ldots,x_n)$, auxiliary input z and security parameter κ, denoted $\mathrm{REAL}_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)$, is defined as the output vector of $\mathsf{P}_1,\ldots,\mathsf{P}_n$ and $\mathsf{A}(z)$ resulting from the protocol*

[13] We assume that a functionality can be computed in polynomial time.

interaction, where for every $i \in \mathcal{I}$, party P_i computes its messages according to A, and for every $j \notin \mathcal{I}$, party P_j computes its messages according to π.

Bounded-parallel composition. An ℓ-times parallel execution of a protocol π' (for a predetermined ℓ) is a protocol π, in which the parties run ℓ independent instances of π' in parallel. Each party receives a vector of ℓ input values, one input for every execution, and all the executions proceed in a synchronous manner, round by round, such that all the messages of the i'th round in all executions of π' are guaranteed to be delivered before round $i+1$ starts. At the conclusion of the executions, every party outputs a vector of ℓ output values, one output value from each execution. Note that the honest parties run each execution of π obliviously to the other executions. (Thus, this is stateless composition.) The adversary may gather information from all the executions in order to attack any specific execution. The ℓ-times parallel execution of π' can also be executed by different subsets of parties, in this case every party that does not participate in the i'th execution receives an empty input λ and outputs λ.

2.3.2 Execution in the Ideal World

In this section, we present standard definitions of ideal-model computations that are used to define security with abort, with identifiable abort, with fairness and with guaranteed output delivery (i.e., full security). We start by presenting the ideal-model computation for security with abort, where the adversary may abort the computation either before or after it has learned the output; other ideal-model computations are defined by restricting the power of the adversary either by forcing the adversary to identify a corrupted party in case of abort, or by allowing fair abort (i.e., abort only before learning the output) or no abort (full security).

Ideal computation with abort. An ideal computation with abort of an n-party functionality f on input $\boldsymbol{x} = (x_1, \ldots, x_n)$ for parties $(\mathsf{P}_1, \ldots, \mathsf{P}_n)$ in the presence of an ideal-model adversary A controlling the parties indexed by $\mathcal{I} \subseteq [n]$, proceeds via the following steps.

Sending inputs to trusted party: An honest party P_i sends its input x_i to the trusted party. The adversary may send to the trusted party arbitrary inputs for the corrupted parties. Let x'_i be the value actually sent as the input of party P_i.

Early abort: The adversary A can abort the computation by sending an abort message to the trusted party. In case of such an abort, the trusted party sends $\perp$ to all parties and halts.

Trusted party answers adversary: The trusted party computes $(y_1, \ldots, y_n) = f(x'_1, \ldots, x'_n)$ and sends y_i to party P_i for every $i \in \mathcal{I}$.

Late abort: The adversary A can abort the computation (*after seeing the outputs of corrupted parties*) by sending an abort message to the trusted party. In case of such abort, the trusted party sends $\perp$ to all parties and halts. Otherwise, the adversary sends a $\mathsf{continue}$ message to the trusted party.

Trusted party answers remaining parties: The trusted party sends y_i to P_i for every $i \notin \mathcal{I}$.

Outputs: Honest parties always output the message received from the trusted party and the corrupted parties output nothing. The adversary A outputs an arbitrary function of the initial

inputs $\{x_i\}_{i\in\mathcal{I}}$, the messages received by the corrupted parties from the trusted party and its auxiliary input.

Definition 2.5 (ideal-model computation with abort)**.** *Let $f\colon (\{0,1\}^*)^n \mapsto (\{0,1\}^*)^n$ be an n-party functionality, let $\mathcal{I} \subseteq [n]$ be the set of indices of the corrupted parties, and let κ be the security parameter. Then, the* joint execution of f under (A, I) in the ideal model*, on input vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, auxiliary input z to A and security parameter κ, denoted $\text{IDEAL}^{\mathsf{abort}}_{f,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x}, \kappa)$, is defined as the output vector of $\mathsf{P}_1, \ldots, \mathsf{P}_n$ and A resulting from the above described ideal process.*

We now define the following variants of this ideal computation:

- **Ideal computation with identifiable abort**. This ideal model proceeds as in Definition 2.5, with the exception that in order to abort the computation, the adversary chooses an index of a corrupted party $i^* \in \mathcal{I}$ and sends (abort, i^*) to the trusted party. In this case the trusted party responds with $(\perp, i^*)$ to all parties. This ideal computation is denoted as $\text{IDEAL}^{\mathsf{id\text{-}abort}}_{f,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x}, \kappa)$.
- **Ideal computation with fairness**. This ideal model proceeds as in Definition 2.5, with the exception that the adversary is allowed to send abort only in step *Early abort*. This ideal computation is denoted as $\text{IDEAL}^{\mathsf{fair}}_{f,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x}, \kappa)$.
- **Ideal computation with fairness and identifiable abort**. This ideal model proceeds as the ideal model for security with fairness, with the exception that in order to abort the computation, the adversary sends (abort, i^*) with $i^* \in \mathcal{I}$ and the trusted party responds with $(\perp, i^*)$ to all parties. This ideal computation is denoted as $\text{IDEAL}^{\mathsf{id\text{-}fair}}_{f,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x}, \kappa)$.
- **Ideal computation with full security (aka guaranteed output delivery)**. This ideal model proceeds as in Definition 2.5, with the exception that the adversary is not allowed to send abort to the trusted party. This ideal computation is denoted as $\text{IDEAL}^{\mathsf{full}}_{f,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x}, \kappa)$.

2.3.3 Security Definitions

Having defined the real and ideal models, we can now define security of protocols according to the real/ideal paradigm.

Definition 2.6. *Let $\mathsf{type} \in \{\mathsf{full}, \mathsf{fair}, \mathsf{id\text{-}fair}, \mathsf{abort}, \mathsf{id\text{-}abort}\}$. Let $f\colon (\{0,1\}^*)^n \mapsto (\{0,1\}^*)^n$ be an n-party functionality, and let π be a probabilistic polynomial-time protocol computing f. The* protocol π (δ, t)-securely computes f with type (and computational security)*, if for every probabilistic polynomial-time real-model adversary A, there exists a probabilistic (expected) polynomial-time adversary S for the ideal model, such that for every $\mathcal{I} \subseteq [n]$ of size at most t, it holds that*

$$\left\{\text{REAL}_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x}, \kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}} \equiv^{\delta}_{c} \left\{\text{IDEAL}^{\mathsf{type}}_{f,\mathcal{I},\mathsf{S}(z)}(\boldsymbol{x}, \kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}}.$$

If δ is negligible, we say that π is a protocol that t-securely computes f with type and computational security.

The protocol π (δ, t)-securely computes f with type (and statistical security)*, if for every real-model adversary A, there exists an adversary S for the ideal model, whose running time is polynomial in the running time of A, such that for every $\mathcal{I} \subseteq [n]$ of size at most t, it holds that*

$$\left\{\text{REAL}_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x}, \kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}} \equiv^{\delta}_{s} \left\{\text{IDEAL}^{\mathsf{type}}_{f,\mathcal{I},\mathsf{S}(z)}(\boldsymbol{x}, \kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}}.$$

If δ is negligible, we say that π is a protocol that t-securely computes f with type *and statistical security.*

Similarly, π is a protocol that (δ, t)-securely computes f with type (and perfect security), *if*

$$\left\{\text{REAL}_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}} \equiv^{\delta} \left\{\text{IDEAL}^{\mathsf{type}}_{f,\mathcal{I},\mathsf{S}(z)}(\boldsymbol{x},\kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}}.$$

If $\delta = 0$, we say that π is a protocol that t-securely computes f with type *and perfect security.*

2.3.4 Reactive Functionalities

The previous section described non-reactive ideal computations (also referred to as secure function evaluation), where each party sends one input to the trusted party and receives back one output; the trusted party is stateless, i.e., no internal state is stored between computations of different functionalities. A reactive functionality is defined as vector of n-party functionalities $\boldsymbol{f} = (f_1, \ldots, f_q)$ where each functionality receives an additional input representing a state. That is, for every $i \in [q]$, the functionality f_i is defined as $(y_1^i, \ldots, y_n^i) = f_i(x_1^i, \ldots, x_n^i, \mathsf{state}^i; r^i)$. In an ideal computation of a reactive functionality $\boldsymbol{f}$ with security type, the trusted party is modeled as a stateful interactive Turing machine. Initially the trusted party sets the internal state to be $\mathsf{state}^1 = \lambda$. Next, the ideal computation proceeds in phases, where in the i'th phase every party and the adversary send their messages to the trusted party according to the non-reactive ideal-model computation of f_i with security type, with the exception that upon receiving inputs $x_1^i, \ldots, x_n^i$ from the honest parties and the adversary, the trusted party samples random coins r^i, computes $(y_1^i, \ldots, y_n^i) = f_i(x_1^i, \ldots, x_n^i, \mathsf{state}^i; r^i)$ and sets $\mathsf{state}^{i+1} = \mathsf{state}^i \circ (x_1^i, \ldots, x_n^i, r^i)$.

Looking ahead, in Section 4.3 (a protocol for Boolean OR) we will make use of a weaker form of reactive functionalities. We define a reactive functionality $\boldsymbol{f} = (f_1, \ldots, f_q)$ to be a single-input reactive functionality, if for every $i \geq 2$, the function f_i is deterministic and does not receive inputs from the parties, i.e., whose output depends only on the state state^2, which is in fact the input values sent for f_1. Stated differently, denote by $x_1, \ldots, x_n$ the input values provided by the parties in the first invocation, then for every $i \in [q]$ the computation is $f_i(x_1, \ldots, x_n)$.

2.3.5 The Hybrid Model

The *hybrid model* is a model that extends the real model with a trusted party that provides ideal computation for specific functionalities. The parties communicate with this trusted party in exactly the same way as in the ideal models described above. The question of which ideal model is considered must be specified. Specifically, the trusted party may work according to any of the ideal models that we have defined above.

Let f be a functionality. Then, an execution of a protocol π computing a functionality g in the f-hybrid model, involves the parties sending normal messages to each other (as in the real model) and in addition, having access to a trusted party computing f. It is essential that the invocations of f are done sequentially, meaning that before an invocation of f begins, the preceding invocation of f must finish. In particular, there is at most a single call to f per round, and no other messages are sent during any round in which f is called. In case f is a reactive functionality then f must be sequentially called until the computation of f is completed.

Let $\mathsf{type} \in \{\mathsf{full}, \mathsf{fair}, \mathsf{id\text{-}fair}, \mathsf{abort}, \mathsf{id\text{-}abort}\}$. Let A be an adversary with auxiliary input z and let $\mathcal{I} \subseteq [n]$ be the set of corrupted parties. We denote by $\text{HYBRID}^{f,\mathsf{type}}_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)$ the random variable

consisting of the view of the adversary and the output of the honest parties, following an execution of π with ideal calls to a trusted party computing f according to the ideal model type, on input vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, auxiliary input z to A, and security parameter κ. We call this the (f, type)-hybrid model.

The following proposition follows from the sequential composition theorem of Canetti [17].

Proposition 2.7. *Let* $\mathsf{type}_1, \mathsf{type}_2 \in \{\mathsf{full}, \mathsf{fair}, \mathsf{id\text{-}fair}, \mathsf{abort}, \mathsf{id\text{-}abort}\}$. *Let* f *be an* n*-party functionality. Let* ρ *be a protocol that* (δ_1, t)*-securely computes* f *with* type_1*, and let* π *be a protocol that* (δ_2, t)*-securely computes* g *with* type_2 *in the* (f, type_1)*-hybrid model, using* q *calls to the ideal functionality. Then protocol* π^ρ*, that is obtained from* π *by replacing all ideal calls to the trusted party computing* f *with the protocol* ρ*, is a protocol that* $((q \cdot \delta_1 + \delta_2), t)$*-securely computes* g *with* type_2 *in the real model.*

2.4 Committee Election

Feige's lightest-bin protocol [31] is an elegant n-party, public-coin protocol, consisting of a single broadcast round, for electing a committee of size $n' < n$, in the information-theoretic setting. Each party uniformly selects one of $\lceil n/n' \rceil$ bins and broadcasts it choice. The parties that selected the lightest bin are elected to participate in the committee. The protocol ensures that the ratio of corrupted parties in the elected committee is similar to their ratio in the population. The original protocol in [31] considered committees of size $\log(n)$; however, this results with a non-negligible failure probability. Boyle et al. [9, Lem. 2.6] analyzed Feige's protocol for arbitrary committee sizes and proved the following lemma.

Lemma 2.8 ([9])**.** *For integers* $n' < n$ *and constants* $0 < \beta < \beta' < 1$*, denote*

$$\mathsf{err}(n, n', \beta, \beta') = \frac{n}{n'} \cdot e^{-\frac{(\beta'-\beta)^2 n'}{2(1-\beta)}}.$$

Feige's lightest-bin protocol is a 1*-round,* n*-party protocol for electing a committee* $\mathcal{C}$*, such that for any set of corrupted parties* $\mathcal{I} \subseteq [n]$ *of size* $t = \beta n$*, the following holds.*

1. $|\mathcal{C}| \leq n'$.
2. $\Pr\left[|\mathcal{C} \setminus \mathcal{I}| \leq (1 - \beta') \cdot n'\right] < \mathsf{err}(n, n', \beta, \beta')$.
3. $\Pr\left[|\mathcal{C} \cap \mathcal{I}| \geq \beta' \cdot |\mathcal{C}|\right] < \mathsf{err}(n, n', \beta, \beta')$.

The committee selected by the lightest-bin protocol of Feige has the desired fraction of honest parties except for probability $\mathsf{err}(n, n', \beta, \beta')$. We capture this guarantee by defining the following ideal functionality.

Definition 2.9 (the committee-election functionality)**.**

Input: *In the committee-election functionality* $f_{\mathsf{elect}}(n, n', \beta')$ *parties have no private input. The common input includes the number of parties* n*, an upper bound* n' *on the size of the elected committee (a subset of parties), and a bound on the fraction of corrupted parties in that committee.*[14]

[14] We note that the functionality is corruption aware, i.e., its actions are depended on knowing the subset $\mathcal{I}$ of corrupted parties. This is standard and does not pose any restrictions since Feige's protocol does not require knowing $\mathcal{I}$.

Computation: *Let $k = n/n'$, the functionality defines k subsets $\mathcal{C}_1, \ldots, \mathcal{C}_k$ (all initially empty). The functionality randomly selects $k \cdot \lceil (1-\beta')n' \rceil$ honest parties and partitions them evenly into the k subsets. Next, the functionality assigns each of the remaining honest parties a random subset $\mathcal{C}_i$. Then, the functionality informs the adversary of the resulting partition.*

Adversary's choice: *The adversary chooses a committee $\mathcal{C}_i$ for some $i \in [k]$ and a subset $\mathcal{J} \subseteq \mathcal{I}$, such that $|\mathcal{C}_i| + |\mathcal{J}| \leq n'$.*

Output: *The output of the functionality is the subset $\mathcal{C} = \mathcal{C}_i \cup \mathcal{J}$.*

The following corollary follows immediately from Lemma 2.8.

Corollary 2.10. *Let $n' < n$ be integers, let $0 < \beta < \beta' < 1$ be constants, and let $t = \beta n$. Then, Feige's lightest-bin protocol $(\mathsf{err}(n, n', \beta, \beta'), t)$-securely computes $f_{\mathsf{elect}}(n, n', \beta')$ with full security. In particular, for $n' = \omega(\log(\kappa))$ Feige's protocol t-securely computes $f_{\mathsf{elect}}(n, n', \beta')$ with full security.*

2.5 Fairness with Restricted Identifiable Abort (with Inputs)

Delegating computation to a small committee will be a useful technique throughout this work. In such a computation, we wish to allow non-members of the committee to monitor the execution of the protocol by committee members; however, non-members should never be able to disrupt the execution themselves. To capture the required security, we introduce a variant of fairness with identifiable abort that will be used as an intermediate step in our constructions.

This definition captures the delegation of the computation to smaller committees that independently carry out the (same) fair computation, such that the adversary can only abort the computation of committees with corrupted parties.

In Section 3, we use this security notion for the case of no-input functionalities. Clearly, this is a special case captured by the general definition. Nevertheless, for clarity, we specify the no-input variant of this notion in Appendix A. We first present a variant of the definition that does not require fairness, which, looking ahead, will turn out to be useful in some of the applications.

Ideal model with restricted identifiable abort. An ideal computation, with $\boldsymbol{\mathcal{C}}$-identifiable-abort, of an n-party functionality f on input $\boldsymbol{x} = (x_1, \ldots, x_n)$ for parties $(\mathsf{P}_1, \ldots, \mathsf{P}_n)$ with respect to $\boldsymbol{\mathcal{C}} = (\mathcal{C}_1, \ldots, \mathcal{C}_\ell)$, where $\mathcal{C}_1, \ldots, \mathcal{C}_\ell \subseteq [n]$, in the presence of an ideal-model adversary A controlling the parties indexed by $\mathcal{I} \subseteq [n]$, proceeds via the following steps.

Sending inputs to trusted party: An honest party P_i sends its input x_i to the trusted party. The adversary may send to the trusted party arbitrary inputs for the corrupted parties. Let x'_i be the value actually sent as the input of party P_i.

Early abort: If there exists a corrupted party in every subset $\mathcal{C}_j$, i.e., if $\mathcal{I} \cap \mathcal{C}_j \neq \emptyset$ for every $j \in [\ell]$, then the adversary A can abort the computation by choosing an index of a corrupted party $i^*_j \in \mathcal{I} \cap \mathcal{C}_j$ for every $j \in [\ell]$ and sending the abort message $(\mathsf{abort}, \{i^*_1, \ldots, i^*_\ell\})$ to the trusted party. In case of such abort, the trusted party sends the message $(\perp, \{i^*_1, \ldots, i^*_\ell\})$ to all parties and halts.

Trusted party answers adversary: If $\mathcal{C}_j \subseteq \mathcal{I}$ for some $j \in [\ell]$, the trusted party sends all the input values $x'_1, \ldots, x'_n$ to the adversary, waits to receive from the adversary output values $y'_1, \ldots, y'_n$, sends y'_i to P_i and proceeds to the *Outputs* step. Otherwise, the trusted party computes $(y_1, \ldots, y_n) = f(x'_1, \ldots, x'_n)$ and sends y_i to party P_i for every $i \in \mathcal{I}$.

Late abort: If there exists a corrupted party in every subset $\mathcal{C}_j$, then the adversary A can abort the computation (*after seeing the outputs of corrupted parties*) by choosing an index $i^*_j \in \mathcal{I} \cap \mathcal{C}_j$ for every $j \in [\ell]$ and sending the abort message $(\mathsf{abort}, \{i^*_1, \ldots, i^*_\ell\})$ to the trusted party. In case of such abort, the trusted party sends the message $(\perp, \{i^*_1, \ldots, i^*_\ell\})$ to all parties and halts. Otherwise, the adversary sends a $\mathsf{continue}$ message to the trusted party.

Trusted party answers remaining parties: The trusted party sends y_i to P_i for every $i \notin \mathcal{I}$.

Outputs: Honest parties always output the message received from the trusted party and the corrupted parties output nothing. The adversary A outputs an arbitrary function of the initial inputs $\{x_i\}_{i \in \mathcal{I}}$, the messages received by the corrupted parties from the trusted party and its auxiliary input.

Definition 2.11 (ideal-model computation with restricted identifiable abort)**.** *Let $f\colon (\{0,1\}^*)^n \mapsto (\{0,1\}^*)^n$ be an n-party functionality, let $\mathcal{I} \subseteq [n]$, and let $\boldsymbol{\mathcal{C}} = (\mathcal{C}_1, \ldots, \mathcal{C}_\ell)$, where $\mathcal{C}_1, \ldots, \mathcal{C}_\ell \subseteq [n]$. The* joint execution of f with $\boldsymbol{\mathcal{C}}$ under (A, I) in the ideal model*, on input vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, auxiliary input z to A, and security parameter κ, denoted $\text{IDEAL}^{\boldsymbol{\mathcal{C}}\text{-id-abort}}_{f,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x}, \kappa)$, is defined as the output vector of $\mathsf{P}_1, \ldots, \mathsf{P}_n$ and $\mathsf{A}(z)$ resulting from the above described ideal process.*

To keep notation short, in case $\boldsymbol{\mathcal{C}} = \{\mathcal{C}_1\}$, i.e., $\ell = 1$, we denote $\mathcal{C}_1$-id-abort instead of $\boldsymbol{\mathcal{C}}$-id-abort. The ideal model presented above defines security with $\boldsymbol{\mathcal{C}}$-identifiable-abort. We define the fair variant of this ideal computation as follows:

Ideal model with fairness and $\boldsymbol{\mathcal{C}}$-identifiable-abort. This ideal model proceeds as in Definition 2.11 with the exception that in step *Late abort*, the adversary is not allowed to abort the computation. This ideal computation is denoted as $\text{IDEAL}^{\boldsymbol{\mathcal{C}}\text{-id-fair}}_{f,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x}, \kappa)$.

Security definitions. We present the security definition according to the ideal model computing f with fairness and $\boldsymbol{\mathcal{C}}$-identifiable-abort. The definitions for security with $\boldsymbol{\mathcal{C}}$-identifiable-abort follow in a similar way.

Definition 2.12. *Let $f\colon (\{0,1\}^*)^n \mapsto (\{0,1\}^*)^n$ be an n-party functionality and let π be a probabilistic polynomial-time protocol computing f. The* protocol π (δ, t)-securely computes f with fairness and (ℓ, n', t')-identifiable-abort (and computational security)*, if for every probabilistic polynomial-time real-model adversary A, there exists a probabilistic polynomial-time adversary S for the ideal model, such that for every $\mathcal{I} \subseteq [n]$ of size at most t and subsets $\mathcal{C}_1, \ldots, \mathcal{C}_\ell \subseteq [n]$ satisfying $|\mathcal{C}_j| = n'$ and $|\mathcal{I} \cap \mathcal{C}_j| \leq t'$, for every $j \in [\ell]$, it holds that*

$$\left\{\text{REAL}_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x}, \kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}} \equiv^{\delta}_{c} \left\{\text{IDEAL}^{(\mathcal{C}_1,\ldots,\mathcal{C}_\ell)\text{-id-fair}}_{f,\mathcal{I},\mathsf{S}(z)}(\boldsymbol{x}, \kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}}.$$

If δ is negligible, we say that π is a protocol that t-securely computes f with fairness and (ℓ, n', t')-identifiable-abort and computational security.

The protocol π (δ, t)-securely computes f with fairness and (ℓ, n', t')-identifiable-abort (and statistical security)*, if for every real-model adversary* A*, there exists an adversary* S *for the ideal model, whose running time is polynomial in the running time of* A*, such that for every* $\mathcal{I} \subseteq [n]$ *of size at most* t*, and subsets* $\mathcal{C}_1, \ldots, \mathcal{C}_\ell \subseteq [n]$ *satisfying* $|\mathcal{C}_j| = n'$ *and* $|\mathcal{I} \cap \mathcal{C}_j| \leq t'$*, for every* $j \in [\ell]$*, it holds that*

$$\left\{\mathrm{REAL}_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}} \equiv_s^{\delta} \left\{\mathrm{IDEAL}^{(\mathcal{C}_1,\ldots,\mathcal{C}_\ell)\text{-id-fair}}_{f,\mathcal{I},\mathsf{S}(z)}(\boldsymbol{x},\kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}}.$$

If δ *is negligible, we say that* π *is a protocol that* t*-securely computes* f *with fairness and* (ℓ, n', t')*-identifiable-abort and statistical security.*

Similarly, π is a protocol that (δ, t)-securely computes f with fairness and (ℓ, n', t')-identifiable-abort (and perfect security)*, if*

$$\left\{\mathrm{REAL}_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}} \equiv^{\delta} \left\{\mathrm{IDEAL}^{(\mathcal{C}_1,\ldots,\mathcal{C}_\ell)\text{-id-fair}}_{f,\mathcal{I},\mathsf{S}(z)}(\boldsymbol{x},\kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}}.$$

If $\delta = 0$*, we say that* π *is a protocol that* t*-securely computes* f *with fairness and* (ℓ, n', t')*-identifiable-abort and perfect security.*

2.5.1 Ideal Functionalities Delegated to Committees

We prove some of our constructions (Constructions 3.7 and 4.8) in a hybrid model that computes (a variant of) the augmented coin-tossing functionality $f_{\mathsf{aug\text{-}ct}}$ (see [32, 18]) and the one-to-many zero-knowledge proof of knowledge functionality $\mathsf{ZK}^{1:\mathrm{M}}$ (see [18, 57]).

Definition 2.13 (delegated augmented coin-tossing)**.** *The augmented coin-tossing functionality, denoted* $f^{\mathcal{C}}_{\mathsf{aug\text{-}ct}}$*, is an* n*-party no-input functionality. It is parametrized by a subset* $\mathcal{C} \subseteq [n]$ *of size* n' *and a commitment scheme* Com*. The output of each party* $\mathsf{P}_i \in \mathcal{C}$ *in this functionality is a private random string* r_i *together with a decommitment information* ρ_i*, and in addition, all* n *parties receive as public output a vector of commitments to the random strings. That is,*

$$f^{\mathcal{C}}_{\mathsf{aug\text{-}ct}}(\lambda,\ldots,\lambda) = ((y_1,\boldsymbol{\sigma}),\ldots,(y_n,\boldsymbol{\sigma})), \text{ where } y_i = \begin{cases} (r_i,\rho_i), & \text{if } \mathsf{P}_i \in \mathcal{C} \\ \lambda, & \text{otherwise} \end{cases}$$

and $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_{n'})$ *such that* $\sigma_j = \mathsf{Com}(r_j; \rho_j)$ *for every* $\mathsf{P}_j \in \mathcal{C}$*.*

Definition 2.14 (one-to-many zero-knowledge)**.** *The one-to-many zero-knowledge proof of knowledge functionality* $\mathsf{ZK}^{1:M}$ *for a language* L *is an* n*-party functionality with a special party, called the prover. The common input of all parties is a statement* x*. The prover sends a statement-witness pair* (x, w) *to the functionality. The common output is* $(x, 1)$ *if* $(x, w) \in R_L$ *and* $(x, 0)$ *otherwise.*

Lemma 2.15. *Assume that TDP and CRH exist and let* $t < n$*. Then,*

1. $\mathsf{ZK}^{1:M}$ *can be* t*-securely computed with full security using a constant-round protocol.*
2. *Let* $\mathcal{C} \subseteq [n]$*, then* $f^{\mathcal{C}}_{\mathsf{aug\text{-}ct}}$ *can be* t*-securely computed with* $\mathcal{C}$*-identifiable-abort using a constant-round protocol.*

Proof. The first part of the lemma follows from Pass [57], who showed that under the assumptions in the lemma there exist constant-round protocols for a simulation-sound,[15] bounded-concurrent composition of $\mathsf{ZK}^{1:\mathrm{M}}$. We consider the variant of the protocol by Beimel et al. [6] that adjusted the protocol from [57], originally designed for an asynchronous point-to-point network, to the synchronous setting where all messages are sent over a broadcast channel.

We now show that $f^{\mathcal{C}}_{\mathsf{aug\text{-}ct}}$ can be securely computed with $\mathcal{C}$-identifiable abort in the $\mathsf{ZK}^{1:\mathrm{M}}$-hybrid model. We use the protocol from Canetti et al. [18] with a small adjustment. In order to obtain a committed random string for $\mathsf{P}'_j \in \mathcal{C}$, all parties in $\mathcal{C}$ proceed as follows.

1. Every party $\mathsf{P}'_i \in \mathcal{C}$ uniformly samples a random string r_{ij}, computes $c_{ij} = \mathsf{Com}(r_{ij}; \rho_{ij})$, and sends $(c_{ij}, (r_{ij}, \rho_{ij}))$ to $\mathsf{ZK}^{1:\mathrm{M}}$ (parameterized with the NP-relation consisting of statement-witness pairs $(c, (r, \rho))$ that satisfy $c = \mathsf{Com}(r; \rho)$). In case $\mathsf{ZK}^{1:\mathrm{M}}$ returns $(c, 0)$ for some $i^* \in \mathcal{C}$, all parties output $(\perp, i^*)$ (for the smallest such i^*) and halt.

2. Every party $\mathsf{P}'_i \in \mathcal{C}$, except for P'_j, broadcasts (r_{ij}, ρ_{ij}), and all parties verify that $c_{ij} = \mathsf{Com}(r_{ij}; \rho_{ij})$; if not the parties output $(\perp, i)$ (for the smallest such i) and halt.

3. Party P'_j computes $r_j = \bigoplus_i r_{ij}$ and outputs $(r_j, (r_{jj}, \rho_{jj}))$ (i.e., the random string is r_j and the decommitment is (r_{jj}, ρ_{jj})), whereas all other parties compute $r^*_j = \bigoplus_{i \neq j} r_{ij}$ and output (c_{jj}, r^*_j) as the commitment for r_j.

The proof of the protocol follows from [18]. □

3 Fairness to Full Security for No-Input Functionalities

In this section, we present a reduction from a fully secure computation to a fair computation for functions without inputs (e.g., coin flipping). This serves as a first step before presenting the more complex case of functionalities with input in Section 4. In Section 3.1, we consider a reduction that does not assume an honest majority, and in Section 3.2 a more efficient reduction in the honest-majority setting. In Section 3.3, we show applications regarding coin-flipping protocols.

Our reductions are two-phased. Initially, we show how to reduce full security to fairness with restricted identifiable abort (defined in Section 2.5, see Appendix A for the no-input case) in a round-efficient manner. Next, we show how to reduce fairness with restricted identifiable abort to fairness.

Recall that if a no-input functionality f has public output, then it can be defined for any number of parties. We denote by f^n the functionality f when defined for n parties, and show how to compile any fair protocol computing $f^{n'}$ to a protocol that fairly computes f^n with restricted identifiable abort (for $n' < n$). For integers $n' < n$ and for $0 < \beta < \beta' < 1$ we define $\mathsf{err}(n, n', \beta, \beta') = \frac{n}{n'} \cdot e^{-\frac{(\beta'-\beta)^2 n'}{2(1-\beta)}}$. In addition, denote by $\mathsf{SS}^{(\mathsf{t},\mathsf{n})}_{\mathsf{out}}(f^n)$ the n-party functionality that computes f^n and outputs shares of the result using a $(t+1)$-out-of-n error-correcting secret-sharing scheme (ECSS, see Definition 2.2). We prove the following theorem.

Theorem 3.1 (restating Theorem 1.1)**.** *Assume that TDP and CRH exist. Let f be a no-input functionality with public output, let $n' < n$ be integers, let $0 < \beta < \beta' < 1$, and let $t = \beta n$ and $t' = \beta' n'$.*

[15] Zero-knowledge protocols are simulation sound if the soundness of each of the protocols is preserved even when the other protocol is simulated at the same time with the roles of the prover and verifier reversed.

1. *If $f^{n'}$ can be (δ', t')-securely computed with fairness by an r'-round protocol, then f^n can be $(t' \cdot \delta' + \mathsf{err}(n, n', \beta, \beta'), t)$-securely computed with full security by an $O(t' \cdot r')$-round protocol.*

2. *If $f^{n'}$ can be (δ', t')-securely computed with full security by an r'-round protocol, then f^n can be $(\delta' + \mathsf{err}(n, n', \beta, \beta'), t)$-securely computed with full security by an $O(t' + r')$-round protocol.*

3. *For $\beta' < 1/2$ and $n' = \min(n, \log(\kappa) \cdot \varphi(\kappa))$ with $\varphi = 1/\sqrt{1-2\beta'} + \Omega(1)$,[16] the following holds unconditionally. If $\mathsf{SS}_{\mathsf{out}}^{(\mathsf{t}', \mathsf{n}')}(f^{n'})$ can be (δ', t')-securely computed with abort by an r'-round protocol, ℓ-times in parallel, for $\ell = \kappa^{\log(e) \cdot \left(\frac{2}{e} + \frac{1}{\varphi(\kappa)}\right)}$, then f^n can be $(\varphi(\kappa)^2 \cdot \ell \cdot \delta' + \mathsf{err}(n, n', \beta, \beta'), t)$-securely computed with full security by an $O(\varphi(\kappa)^2 \cdot r')$-round protocol.*

The proof of Theorem 3.1 is given in the sections below, where the first part follows from a combination of Theorems 3.2 and 3.4; the second part from Theorem 3.2 and Corollary 3.9; and the third part from Theorems 3.10 and 3.14.

3.1 Fairness to Full Security without an Honest Majority (no Inputs)

We now present a reduction from full security to fairness for no-input functionalities when an honest majority is not assumed. In Section 3.1.1, we show how to compute f^n with full security in the hybrid model computing f^n with fairness and restricted identifiable abort. In Section 3.1.2, we show how to compile a fair protocol for $f^{n'}$ to a fair protocol for f^n with restricted identifiable abort.

3.1.1 Fairness with Restricted Identifiable Abort to Full Security

We start by showing how to reduce full security to fairness with restricted identifiable abort. A single committee $\mathcal{C}$ is considered in this setting (i.e., $\ell = 1$). The idea is quite simple: initially, a committee $\mathcal{C}$ is elected using Feige's lightest-bin protocol [31] such that the ratio of corrupted parties in the committee is approximately the same as in the original party-set (see Section 2.4). Next, the parties sequentially call the fair computation with $\mathcal{C}$-identifiable-abort (where only corrupted parties in $\mathcal{C}$ can abort the computation, in which case they are identified by all parties), until receiving the output.

Theorem 3.2. *Let f be a no-input, n-party functionality with public output, let $n' < n$, let $0 < \beta < \beta' < 1$, and let $t = \beta n$ and $t' = \beta' n'$. Then, f can be $(\mathsf{err}(n, n', \beta, \beta'), t)$-securely computed with full security in a hybrid model that computes f with fairness and (n', t')-identifiable-abort, by using $t' + 1$ sequential calls to the ideal functionality.*

Proof. We present the protocol in the hybrid model that computes the committee-election functionality f_{elect} with full security (f_{elect} is defined in Section 2.4).

Protocol 3.3. *(fairness with restricted identifiable abort to full security (no inputs))*

- ***Hybrid Model:*** *The protocol is defined in the hybrid model computing f_{elect} with full security, and f with fairness and (n', t')-identifiable-abort.*
- ***Common Input:*** *The values $t', n' \in \mathbb{N}$.*

[16] By $\varphi = 1/\sqrt{1-2\beta'} + \Omega(1)$ we mean that for sufficiently large κ it holds that $\varphi(\kappa) > 1/\sqrt{1-2\beta'}$.

- ***The Protocol:***

1. *All the parties invoke* $f_{\mathsf{elect}}(n, n', \beta')$ *and elect a committee* $\mathcal{C}_1 \subset [n]$ *of size* n'.
2. *For* $i = 1, \ldots, t' + 1$ *do*
   (a) *All parties that have not been previously identified call the trusted party computing* $(f, \mathcal{C}_1\text{-}\mathsf{id\text{-}fair})$*, where the party with the lowest index in* $\mathcal{C}_i$ *simulates all parties in* $\mathcal{C}_1 \setminus \mathcal{C}_i$*. Denote the output* P_j *receives by* y_j.
   (b) *Every party* P_j *checks if* y_j *is a valid output, if so* P_j *outputs* y_j *and halts. Otherwise, all parties received* $(\perp, i^*)$ *as output, where* $i^* \in \mathcal{C}_1 \cap \mathcal{I}$*. If* $i^* \notin \mathcal{C}_i$ *(and so* P_{i^*} *is a previously identified corrupted party), then all parties set* i^* *to be the lowest index in* $\mathcal{C}_i$.
   (c) *All parties set* $\mathcal{C}_{i+1} = \mathcal{C}_i \setminus \{i^*\}$.

Let A be an adversary attacking Protocol 3.3 and let $\mathcal{I}$ be the set of corrupted parties. We construct a simulator S for the ideal model computing f with full security, as follows. S starts by emulating A on its auxiliary input z. Initially, S emulates the committee-election functionality $f_{\mathsf{elect}}(n, n', \beta')$; that is, S partitions the honest parties to n/n' random subsets, each containing at least $(1 - \beta') \cdot n$ parties, hands them to A and receives back a committee $\mathcal{C}_1$ of size n' that contains exactly one of the subsets (and does not intersect any of the others).

Next, in every iteration, S simulates toward A the computation of f with $\mathcal{C}_1$-identifiable-abort. If A sends the message (abort, i^*) in the i'th iteration, with $i^* \in \mathcal{I} \cap \mathcal{C}_1$, then S checks if P_{i^*} has not been previously identified (otherwise set i^* to be the smallest element in $\mathcal{C}_i$), simulates sending the response $(\perp, i^*)$ to all parties, sets $\mathcal{C}_{i+1} = \mathcal{C}_i \setminus \{i^*\}$ and proceeds to the next iteration. In the first iteration in which no abort is sent, S calls the trusted party in the fully secure ideal model computing f. Upon receiving the output from its trusted party, S hands it to A as if it were the output of the corrupted parties in the iteration of π, and outputs whatever A outputs.

The simulation in the hybrid model is perfect since S can perfectly simulate the trusted party for all iterations in which abort is sent. Furthermore, in the first iteration for which abort is not sent, S sends to A the output of the function f as computed in the protocol. After t' iterations it is guaranteed that there are no corrupted parties left in $\mathcal{C}_1$ and the protocol will complete. By Corollary 2.10, Feige's lightest-bin protocol $(\mathsf{err}(n, n', \beta, \beta'), t)$-securely computes $f_{\mathsf{elect}}(n, n', \beta')$ with full security and the theorem follows. □

3.1.2 Fairness to Fairness with Restricted Identifiable Abort

We next present a reduction from a fair computation with restricted identifiable abort of f^n to a fair computation of $f^{n'}$. More specifically, let π' be a fair protocol computing $f^{n'}$ by a subset of n' parties $\mathcal{C}$. We show that π' can be compiled into a protocol π that computes f with fairness and $\mathcal{C}$-identifiable-abort. The underlying idea is to let the committee $\mathcal{C}$ prove that every step in the execution is correct (in a similar way to the GMW compiler [33]) such that when π' terminates the parties in $\mathcal{C}'$ either obtain the output or identify a corrupted party. Next, every party in the committee broadcasts the result and proves that it is indeed the correct result to all n parties.

The above is formally stated in the theorem below, using the following notations. Let f be a no-input functionality with public output, let $t, n' < n$, let $t' < n'$, and let $\mathcal{C} \subseteq [n]$ of size n'.

Theorem 3.4. *Assume that TDP and CRH exist, and let f be a no–input functionality with public output. Then, there exists a* PPT *algorithm* $\mathsf{Compiler}^{n'\to n}_{\text{no-in}}$ *such that for any n'-party, r'-round protocol π' computing $f^{n'}$, the protocol $\pi = \mathsf{Compiler}^{n'\to n}_{\text{no-in}}(\pi', \mathcal{C})$ is an n-party, $O(r')$-round protocol computing f^n with the following guarantee. If the number of corrupted parties in $\mathcal{C}$ is at most t', and π' is a protocol that (δ', t')-securely computes $f^{n'}$ with fairness, then π is a protocol that (δ', t)-securely computes f^n with fairness and $\mathcal{C}$-identifiable-abort.*

Proof. We construct (in Construction 3.7) the protocol compiler $\mathsf{Compiler}^{n\to n'}_{\text{no-in}}(\pi', \mathcal{C})$ in the $(f_{\mathsf{aug\text{-}ct}}, \mathsf{ZK}^{1:\mathrm{M}})$-hybrid model (see Definitions 2.13 and 2.14), and prove the security properties of this compiler in Lemma 3.8. Lemma 2.15, proves that the ideal functionalities $f_{\mathsf{aug\text{-}ct}}$ and $\mathsf{ZK}^{1:\mathrm{M}}$ can be instantiated in a round-preserving manner. The proof follows from the sequential composition theorem (Proposition 2.7). □

Compiling a fair protocol to a protocol with fairness and restricted identifiable abort. Toward proving Theorem 3.4, we next describe the compiler from fairness to fairness and restricted identifiable abort. We begin by making two remarks regarding the construction.

Remark 3.5 (fairness to fairness with identifiable abort)**.** *The compiler described in Construction 3.7 assumes that the protocol π' (the input of the compiler) is fair with identifiable abort and that all communication in the protocol π' is sent over the broadcast channel. These assumptions are without loss of generality. Following [22, Lem. 3] any fair protocol can be compiled in a round-preserving manner into a protocol that provides fairness with identifiable abort, tolerating any number of corrupted parties, in the $(f_{\mathsf{aug\text{-}ct}}, \mathsf{ZK}^{1:M})$-hybrid model.*[17]

Remark 3.6 (Using functionality $\mathsf{ZK}^{1:\mathrm{M}}$ in the compiler)**.** *We will consider the relation R_j (for the j'th party in $\mathcal{C}$), parametrized by a commitment scheme* Com*, that contains pairs $((\boldsymbol{m}, \sigma, \mathsf{out}), (r, \rho))$, and validates that* out *is the output value of protocol π' using randomness r and messages $\boldsymbol{m} = (m_1, \ldots, m_p)$, and that $\sigma = \mathsf{Com}(r; \rho)$.*

The n-party protocol $\pi = \mathsf{Compiler}^{n'\to n}_{\text{no-in}}(\pi', \mathcal{C})$ is defined as follows.

Construction 3.7. *(fairness to fairness with restricted identifiable abort)*

- ***Hybrid Model:*** *The protocol is defined in the hybrid model computing $f_{\mathsf{aug\text{-}ct}}$ with restricted identifiable abort and $\mathsf{ZK}^{1:M}$ with full security.*
- ***Common Input:*** *A subset $\mathcal{C} \subseteq [n]$ of size n' and an n'-party protocol π', computing the functionality $f^{n'}$ with fairness and identifiable abort, using only a broadcast channel (see Remark 3.5). We use the notation P'_j to refer to the j'th party in $\mathcal{C}$.*
- ***The Protocol:***

1. *All parties invoke $f^{\mathcal{C}}_{\mathsf{aug\text{-}ct}}$ with $\mathcal{C}$-identifiable-abort, every $\mathsf{P}'_j \in \mathcal{C}$ receives back $(r_j, \rho_j, \boldsymbol{\sigma})$ where $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_{n'})$ is common to all n parties. In case the computation for $\mathcal{C}$ aborts with the identity of party $\mathsf{P}_{i^*} = \mathsf{P}'_{j^*} \in \mathcal{C}$, all n parties output $(\perp, i^*)$ and halt.*

[17] The original proof in [22] assumes one-way functions and is not round preserving, however, using the techniques from [57], the compilation will blow-up the round complexity only by a constant factor.

2. *The parties in $\mathcal{C}$ execute the protocol π' for computing $f^{n'}$ over the broadcast channel, where P'_j uses r_j as its random coins. Let out_j be the output P'_j received (either a valid value y or $(\perp, i^*)$ with $\mathsf{P}_{i^*} = \mathsf{P}'_{j^*} \in \mathcal{C}$). Denote by $\boldsymbol{m}_j = (m_1^j, \ldots, m_p^j)$ the messages P'_j received during the protocol.*

3. *Every $\mathsf{P}'_j \in \mathcal{C}$ invokes $\mathsf{ZK}^{1:M}$ and proves to all parties that out_j is indeed the correct output value generated by π' using the committed randomness r_j and messages $\boldsymbol{m}_j$, i.e., P'_j sends $((\boldsymbol{m}_j, \sigma_j, \mathsf{out}_j), (r_j, \rho_j))$ to $\mathsf{ZK}^{1:M}$ (parametrized by the relation R_j). Once party P_i receives from $\mathsf{ZK}^{1:M}$ an accepting proof for P'_j's output value out_j, party P_i outputs this value (invalid proofs are ignored).*

Lemma 3.8. *Assume that commitment schemes exist and consider the same notations as in Theorem 3.4. If π' is a (δ', t')-secure protocol computing $f^{n'}$ with fairness, then the protocol $\pi = \mathsf{Compiler}^{n' \to n}_{\mathsf{no\text{-}in}}(\pi', \mathcal{C})$ is an n-party protocol that (δ', t)-securely computes f^n with fairness and $\mathcal{C}$-identifiable-abort, in the $(f_{\mathsf{aug\text{-}ct}}, \mathsf{ZK}^{1:M})$-hybrid model.*

Proof. Let A be an adversary attacking the execution of protocol π in the $(f_{\mathsf{aug\text{-}ct}}, \mathsf{ZK}^{1:\mathrm{M}})$-hybrid model and let $\mathcal{I} \subseteq [n]$ be a subset of size at most t, satisfying $|\mathcal{I} \cap \mathcal{C}| \leq t'$. We construct the following adversary S for the ideal model computing f with fairness and $\boldsymbol{\mathcal{C}}$-identifiable-abort. S starts by emulating A on the auxiliary input z. The simulator interacts with A, playing the roles of the honest parties and the ideal functionalities $f_{\mathsf{aug\text{-}ct}}$ and $\mathsf{ZK}^{1:\mathrm{M}}$.

To simulate Step 1, the simulator S plays $f^{\mathcal{C}}_{\mathsf{aug\text{-}ct}}$ honestly by sampling random strings (r_j, ρ_j) for parties $\mathsf{P}'_j \in \mathcal{C}$, computing $\sigma_j = \mathsf{Com}(r_j; \rho_j)$ and setting $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_{n'})$. Next, S hands $(r_j, \rho_j, \boldsymbol{\sigma})$ to corrupted parties $\mathsf{P}'_j \in \mathcal{C}$ and $(\lambda, \boldsymbol{\sigma})$ to corrupted parties outside of $\mathcal{C}$. In case S receives (abort, i^*) with $i^* \in \mathcal{I} \cap \mathcal{C}$ from A, it forwards (abort, i^*) to the trusted party, responds with $(\perp, i^*)$ to A, outputs whatever A outputs and halts.

Next, the simulator S uses the simulator $\tilde{\mathsf{S}}$ that is guaranteed to exist for π' when interacting with the residual adversary of A in Step 2 (i.e., the adversary against the protocol π' that is induced from the behavior of A). The simulator S invokes $\tilde{\mathsf{S}}$ on auxiliary information containing z and the view of the adversary in the simulation until this point. If $\tilde{\mathsf{S}}$ sends (abort, i^*) with $i^* \in \mathcal{I} \cap \mathcal{C}$, the simulator forwards (abort, i^*) to the trusted party and sets $\mathsf{out} = (\perp, i^*)$. Otherwise, S sends empty strings as the input of the corrupted parties, receives back the output y and sets $\mathsf{out} = y$. Next, The simulator S forwards y to $\tilde{\mathsf{S}}$ as the output of the computation, receives back the output from $\tilde{\mathsf{S}}$, which contains the view of the adversary, and interacts with A accordingly.

To simulate Step 3, the simulator S simulates $\mathsf{ZK}^{1:\mathrm{M}}$. The simulator sends, on behalf of every honest party $\mathsf{P}'_j \in \mathcal{C}$, the message $((\boldsymbol{m}_j, \sigma_j, \mathsf{out}), 1)$ to every corrupted party, where $\boldsymbol{m}_j$ is obtained from the output of $\tilde{\mathsf{S}}$. In addition, S receives $((\boldsymbol{m}'_j, \sigma'_j, \mathsf{out}'_j), (r'_j, \rho'_j))$ from A on behalf of every corrupted party $\mathsf{P}'_j \in \mathcal{C}$ and verifies according to the relation R_j (incorrect proofs are ignored). Finally, S outputs whatever A outputs and halts.

Computational indistinguishability between the real execution of the compiled protocol π running with adversary A and the ideal computation of f running with S follows directly from the security of π'. □

The proof of Theorem 3.4 can be easily adjusted to the case where π' is a fully secure protocol for computing $f^{n'}$. In this case, since the augmented coin-tossing functionality $f^{\mathcal{C}}_{\mathsf{aug\text{-}ct}}$ is secure with $\mathcal{C}$-identifiable-abort (see Section 2.5.1), the adversary can force to restart it $t'+1$ times. Once $f^{\mathcal{C}}_{\mathsf{aug\text{-}ct}}$ completes, the adversary cannot abort the computation. This yields the following corollary.

Corollary 3.9. *Assume that TDP and CRH exist. Then, there exists a* PPT *algorithm* $\mathsf{Compiler}_{\text{no-in}}^{n' \to n}$ *such that for any* n'*-party,* r'*-round protocol* π' *computing* $f^{n'}$*, the protocol* $\pi = \mathsf{Compiler}_{\text{no-in}}^{n' \to n}(\pi', \mathcal{C})$ *is an* n*-party,* $O(t' + r')$*-round protocol computing* f^n *with the following guarantee. If the number of corrupted parties in* $\mathcal{C}$ *is at most* t'*, and* π' *is a protocol that* (δ', t')*-securely computes* $f^{n'}$ *with full security, then* π *is a protocol that* (δ', t)*-securely computes* f^n *with full security.*

3.2 Fairness to Full Security with an Honest Majority (no Inputs)

We now turn to the honest-majority setting, and present a reduction from full security to fairness for no-input functionalities. In Section 3.2.1, we show how to compute f^n with full security in the hybrid model computing f^n with fairness with restricted identifiable abort using $\Omega(1)$ calls. In Section 3.2.2, we show how to compile a fair protocol for $f^{n'}$ to a fair protocol for f^n with restricted identifiable abort, unconditionally, and in a black-box manner.

3.2.1 Reducing the Round Complexity with an Honest Majority

In the honest-majority setting, we are able to utilize parallel computations in many committees in order to reduce the number of calls to the ideal computation from $\Omega(\log(\kappa))$ to $\Omega(1)$. The idea is to start as in the no-honest-majority setting (Protocol 3.3) by electing a (super-)logarithmic committee $\mathcal{C}$, but instead of sequentially invoking the computation with $\mathcal{C}$-identifiable-abort, we consider multiple sub-committees $\boldsymbol{\mathcal{C}} = (\mathcal{C}_1, \ldots, \mathcal{C}_\ell)$, where $\mathcal{C}_j \subseteq \mathcal{C}$, and invoke a computation with $\boldsymbol{\mathcal{C}}$-identifiable-abort. By defining the subsets in $\boldsymbol{\mathcal{C}}$ appropriately, we can ensure that every sub-committee has an honest majority, and that the adversary must reveal the identity of many corrupted parties in order to abort the computation.

Theorem 3.10. *Let* f *be a no-input functionality with public output, let* $0 < \beta < \beta' < 1/2$*, let* $n \in \mathbb{N}$*, let* $n' = \log(\kappa) \cdot (\varphi(\kappa) - 1/\varphi(\kappa))$ *with* $\varphi = 1/\sqrt{1 - 2\beta'} + \Omega(1)$*, let* $\ell = \kappa^{\log(e) \cdot \left(\frac{2}{e} + \frac{1}{\varphi(\kappa)}\right)}$*, and let* $t = \beta n$ *and* $t' = \beta' n'$*. Then,* f^n *can be* $(\mathsf{err}(n, n', \beta, \beta'), t)$*-securely computed with full security in a hybrid model that computes* f^n *with fairness and* (ℓ, n', t')*-identifiable-abort, by invoking the ideal functionality in* $\varphi(\kappa)^2$ *rounds.*

Proof. Assume for simplicity of exposition that $n = \Omega(\log(\kappa) \cdot \varphi(\kappa))$. We modify Protocol 3.3 as follows. First, the parties elect a committee $\mathcal{C} \subseteq [n]$ of size $m = \log(\kappa) \cdot \varphi(\kappa)$ by invoking $f_{\mathsf{elect}}(n, m, \beta')$. Consider all subsets of $\mathcal{C}$ of size $n' = m - n''$ for $n'' = \log(\kappa)/\varphi(\kappa)$, denoted $\boldsymbol{\mathcal{C}} = (\mathcal{C}_1, \ldots, \mathcal{C}_{\ell'})$ with $\ell' = \binom{m}{n'}$. Next, the parties proceed in iterations, where in each iteration they invoke the ideal functionality computing f with fairness and $\boldsymbol{\mathcal{C}}$-identifiable-abort. In case the output value is a subset of identified corrupted parties, they are removed and another iteration is carried out. Otherwise, the parties halt with the output value.

We start by showing that the protocol indeed runs in polynomial time, i.e., that the number of sub-committees is polynomial.

Claim 3.11. $\binom{m}{n'} \leq n^{\log e \cdot \left(\frac{2}{e} + \frac{1}{\varphi(\kappa)}\right)}$.

Proof. For every $1 < k < m$ it holds that $\binom{m}{k} \leq (em/k)^k$, therefore,

$$\begin{aligned}\binom{m}{n'} = \binom{m}{n''} \leq \left(\frac{em}{n''}\right)^{n''} &= \left(\frac{e \cdot \log(\kappa) \cdot \varphi(\kappa)}{\log(\kappa)/\varphi(\kappa)}\right)^{\log(\kappa)/\varphi(\kappa)} \\ &= \left(e \cdot \varphi(\kappa)^2\right)^{\log(\kappa)/\varphi(\kappa)} \\ &= 2^{\frac{\log\left(e \cdot \varphi(\kappa)^2\right) \cdot \log(\kappa)}{\varphi(\kappa)}} \\ &= \kappa^{\left(\frac{\log e}{\varphi(\kappa)} + \frac{2\log\varphi(\kappa)}{\varphi(\kappa)}\right)} \\ &\overset{(*)}{\leq} \kappa^{\left(\frac{\log e}{\varphi(\kappa)} + \frac{2\log e}{e}\right)},\end{aligned}$$

where $(*)$ follows by Steiner [60] who showed that $x^{1/x}$ is bounded from above by $e^{1/e}$, for every positive x, hence

$$\frac{\log\varphi(\kappa)}{\varphi(\kappa)} = \log\left(\varphi(\kappa)^{1/\varphi(\kappa)}\right) \leq \log\left(e^{1/e}\right) = \frac{\log e}{e}.$$

□

Next, we show that every sub-committee still has an honest majority.

Claim 3.12. *In every sub-committee $\mathcal{C}_i$ there exists an honest majority.*

Proof. We require that $n' > 2 \cdot |\mathcal{C} \cap \mathcal{I}|$. By the definition of $f_{\mathsf{elect}}(n, m, \beta')$ it holds that $|\mathcal{C} \cap \mathcal{I}| \leq \beta' \cdot m$. The claim will therefore follow if $n' > 2 \cdot \beta' \cdot m$, i.e.,

$$\log(\kappa) \cdot (\varphi(\kappa) - 1/\varphi(\kappa)) > 2 \cdot \beta' \cdot \log(\kappa) \cdot \varphi(\kappa). \tag{1}$$

Since $0 < \beta' < 1/2$ and $\varphi = \Omega(1)$, Equation (1) holds for

$$\varphi(\kappa) > \frac{1}{\sqrt{1 - 2\beta'}}.$$

□

Finally, we show that after calling the ideal functionality in $\varphi(\kappa)^2$ rounds, the adversary cannot abort the computation.

Claim 3.13. *After $\varphi(\kappa)^2$ iterations, the protocol terminates.*

Proof. Denote by $\mathcal{J}$ the set of identified corrupted parties in some iteration of the protocol. If $|\mathcal{J}| < n''$, then $|\mathcal{C} \setminus \mathcal{J}| > m - n''$; hence, there exists a sub-committee $\mathcal{C}_i \subseteq \mathcal{C} \setminus \mathcal{J}$ of size $m - n''$ such that the adversary didn't identify any corrupted parties from $\mathcal{C}_i$. It follows that in every iteration either the computation completes or at least n'' corrupted parties are identified. Therefore, after $m/n'' = \varphi(\kappa)^2$ iterations, the computation will complete. □

The simulator proceeds similarly to the simulator in Theorem 3.2, where in every iteration, if the adversary sends $(\mathsf{abort}, \mathcal{J})$, with $\mathcal{J} \subseteq \mathcal{I}$ and $\mathcal{J} \cap \mathcal{C}_l \neq \emptyset$ for every $l \in [\ell']$, the simulator S simulates sending $(\perp, \mathcal{J})$ to the parties. If A does not send abort, S calls the trusted party in the fully secure ideal model computing f, obtains the output y, forwards y to the adversary, and outputs whatever A outputs. □

3.2.2 Fairness with Restricted Identifiable Abort with an Honest Majority

Now, we show that in the honest-majority setting, the reduction from fairness with restricted identifiable abort to fairness can be much more elegant, and more importantly, be based on much simpler tools. Specifically, we devise a compiler, similar to the one for the no-honest-majority case, that is based solely on error-correcting secret sharing schemes (ECSS), which exist unconditionally (see Definition 2.2).

Given a no-input, n-party functionality with public output, we consider the no-input, n'-party functionality, denoted $\mathsf{SS}_{\mathsf{out}}^{(\mathsf{t}',\mathsf{n}')}(f)$ that computes f^n and outputs shares of the result using $(t'+1)$-out-of-n' ECSS, for some $t' < n/2$. Note that a secure computation of $\mathsf{SS}_{\mathsf{out}}^{(\mathsf{t}',\mathsf{n}')}(f)$ with abort is in fact fair assuming t' corruptions, since an adversary aborting the computation does not learn any new information.

We prove the theorem below, using the following notations. Let f be a no-input functionality with public output, let $t < n/2$, let $n' < n$, let $t' < n'/2$, and let $\boldsymbol{\mathcal{C}} = (\mathcal{C}_1, \ldots, \mathcal{C}_\ell)$, where $\mathcal{C}_l \subseteq [n]$ for every $l \in [\ell]$.

Theorem 3.14. *There exists a* PPT *algorithm* $\mathsf{Compiler}_{\mathsf{hm,\ no\text{-}in}}^{n'\to n}$ *such that if the number of corrupted parties in every* $\mathcal{C}_j$ *is at most* t'*, then the following holds with information-theoretic security. For any* n'*-party,* r'*-round protocol that* (δ', t')*-securely computes* $\mathsf{SS}_{\mathsf{out}}^{(\mathsf{t}',\mathsf{n}')}(f)$ *with abort,* ℓ *times in parallel, the protocol* $\pi = \mathsf{Compiler}_{\mathsf{hm,\ no\text{-}in}}^{n'\to n}(\pi', \boldsymbol{\mathcal{C}})$ *is an* n*-party,* $O(r')$*-round protocol that* $(\ell \cdot \delta', t)$*-securely computes* f *with fairness and* $\boldsymbol{\mathcal{C}}$*-identifiable-abort. Furthermore, the compiler is black-box with respect to the protocol* π'*.*

Proof. In Lemma 3.15, we prove the theorem under the assumption that π' is secure with identifiable abort. In Lemma 3.17, we explain how to compile, in a round-preserving manner, every protocol that is secure with abort into a protocol that is secure with identifiable abort in the honest-majority setting, with information-theoretic security and using only a black-box access to the underlying protocol. □

Lemma 3.15. *Consider the same notations as in Theorem 3.14. If* π' *is protocol that* (δ', t')*-securely computes* $\mathsf{SS}_{\mathsf{out}}^{(\mathsf{t}',\mathsf{n}')}(f)$ *with identifiable abort,* ℓ *times in parallel, then the protocol* $\pi = \mathsf{Compiler}_{\mathsf{hm,\ no\text{-}in}}^{n'\to n}(\pi', \boldsymbol{\mathcal{C}})$ *is an* n*-party protocol that* $(\ell \cdot \delta', t)$*-securely computes* f^n *with fairness and* $\boldsymbol{\mathcal{C}}$*-identifiable-abort.*

Proof. Given the protocol π' and $\boldsymbol{\mathcal{C}} = (\mathcal{C}_1, \ldots, \mathcal{C}_\ell)$, the n-party protocol $\pi = \mathsf{Compiler}_{\mathsf{hm,\ no\text{-}in}}^{n'\to n}(\pi', \boldsymbol{\mathcal{C}})$ is defined as follows.

Construction 3.16. *(security with abort to fairness with restricted identifiable abort)*

- ***Common Input:*** *An* n'*-party protocol* π' *and* $\boldsymbol{\mathcal{C}} = (\mathcal{C}_1, \ldots, \mathcal{C}_\ell)$*, where each* $\mathcal{C}_l \subseteq [n]$ *is of size* n'*. We use the notation* P_j^l *to refer to the* j*'th party in* $\mathcal{C}_l$*.*
- ***The Protocol:***

1. *For every committee* $\mathcal{C}_l$*, the parties in* $\mathcal{C}_l$ *execute the protocol* π' *for computing* $\mathsf{SS}_{\mathsf{out}}^{(\mathsf{t}',\mathsf{n}')}(f)$ *with identifiable abort. Let* $\mathsf{out}_{j,l}$ *be the output* $\mathsf{P}_j^l \in \mathcal{C}_l$ *received (either a valid share* y_j^l *or* $(\perp, i^*)$ *with* $\mathsf{P}_{i^*} = \mathsf{P}_{j^*}^l \in \mathcal{C}_l$*). If* $\mathsf{out}_{j,l}$ *is of the form* $(\perp, i^*)$*, then* P_j^l *broadcasts* $(l, \mathsf{out}_{j,l})$*.*

2. *If for every committee $\mathcal{C}_l$, more than t' parties broadcasted a value, then every party P_i takes the value (l, i_l^*) that appears the most for every $\mathcal{C}_l$ (if there is no unique majority value, choose arbitrarily) and outputs $(\perp, \{i_1^*, \ldots, i_\ell^*\})$. Otherwise, denote by l^* the minimal index l such that at most t' values (l, i^*) were broadcasted.*

3. *Every party $\mathsf{P}_j^{l^*} \in \mathcal{C}_{l^*}$ broadcasts its output value out_{j,l^*}.*

4. *Denote by $y_j^{l^*}$ the value broadcasted by $\mathsf{P}_j^{l^*}$. Every party computes $y = \mathsf{Recon}(y_1^{l^*}, \ldots, y_{n'}^{l^*})$ and outputs y.*

Let A be a computationally unbounded adversary attacking the execution of π, and let $\mathcal{I} \subseteq [n]$ satisfying $|\mathcal{I} \cap \mathcal{C}_l| \leq t'$ for every $l \in [\ell]$. We construct an adversary S (simulator) for the ideal model computing f with fairness and $\boldsymbol{\mathcal{C}}$-identifiable-abort. The simulator S starts by emulating A on the auxiliary input z. The simulator S interacts with A, playing the roles of the honest parties. For simplicity, assume that the output value of f are elements in $\{0,1\}^\kappa$.

To simulate Step 1, the simulator uses the simulator $\tilde{\mathsf{S}}$ that is guaranteed to exist for the ℓ-times parallel execution of π' with the residual adversary of A in Step 1. S invokes $\tilde{\mathsf{S}}$ on the auxiliary z and receives from $\tilde{\mathsf{S}}$ early messages (abort, i_l^*) for the l'th computation. If $\tilde{\mathsf{S}}$ did not abort the computation of a committee $\mathcal{C}_l$, the simulator S hands secret shares of 0^κ to the corrupted parties in $\mathcal{C}_l \cap \mathcal{I}$. Next, $\tilde{\mathsf{S}}$ may send late-abort messages (abort, i_l^*). Finally, the simulator S receives the output from $\tilde{\mathsf{S}}$ that contains the simulated view of the adversary, and interacts with A accordingly.

To simulate Step 2, for every $\mathcal{C}_l$ for which $\tilde{\mathsf{S}}$ aborted the computation, S simulates broadcasting (l, i_l^*) by all honest parties in $\mathcal{C}_l$. In case $\tilde{\mathsf{S}}$ aborted the computations for every committee $\mathcal{C}_l$, the simulator S sends $(\mathsf{abort}, \{i_1^*, \ldots, i_\ell^*\})$ to the trusted party (in the ideal model with fairness and $\boldsymbol{\mathcal{C}}$-identifiable-abort). Otherwise, S receives the output y from the trusted party, secret shares y and simulates the honest parties in $\mathcal{C}_{l^*}$ broadcasting their shares, for simulating Step 3. Finally, S outputs whatever A outputs and halts.

The proof follows in a straightforward manner based on the unconditional security of the ECSS scheme and of the protocol π'. □

Lemma 3.17. *Let f be an n-party functionality and let $t < n/2$. The following holds with information-theoretic security. There exists a* PPT *compiler C such that if π is an r-round protocol that (δ, t)-securely computes f with abort, then $C(\pi)$ is an $O(r)$-round protocol that (δ, t)-securely computes f with identifiable abort. Moreover, the compiler uses the protocol π in a black-box manner.*

Proof. In the proof, we combine existing results from [45, 23]. We start by using the compiler of Ishai et al. [45, Thm. 6], from security with abort to security with identifiable abort, that satisfies the requirements in the lemma, in the Setup-Commit-then-Prove hybrid model.[18] In [45], the Setup-Commit-then-Prove functionality was realized in a hybrid model that gave correlated randomness to the parties. Namely, they proved the following claim.

Claim 3.18. *Let π be an r-round protocol which (δ, t)-securely computes f with abort and with information-theoretic security in the correlated-randomness model for a distribution D (i.e., where*

[18] The Setup-Commit-then-Prove is a reactive functionality, parametrized by a vector of NP-relations, that in the first invocation hands every party a secret witness, and in all future invocation allows each party to prove NP-statements to all other parties, using the secret witness. This functionality extends the Commit-then-Prove functionality from [18].

a trusted dealer samples correlated randomness for the parties in the setup phase). There exists a PPT *compiler C such that $C(\pi)$ is an $O(r)$-round protocol that (δ, t)-securely computes f with identifiable abort in the Setup-Commit-then-Prove hybrid model and the correlated randomness model. Moreover, the compiler uses the protocol π in a black-box manner.*

Cohen et al. [23, Lem. 6.2] showed how to security compute (without abort) the required correlated randomness in the honest-majority setting by a constant-round protocol. Namely, they proved the following claim.

Claim 3.19. *Let π be a constant-round protocol which (δ, t)-securely computes f with identifiable abort and with information-theoretic security in the Setup-Commit-then-Prove hybrid model and the correlated-randomness model for an efficiently sampleable distribution D in NC^0. Then, f can be (δ, t)-securely computed with identifiable abort and with information-theoretic in constant rounds in the broadcast model with secure point-to-point channels.*

The lemma follows from these results. □

3.3 Applications

We next give a few applications of the above security uplifting reductions with respect to coin-flipping protocols.

Definition 3.20. *The n-party coin-flipping functionality is defined as $f^n_{\mathsf{cf}}(\lambda, \ldots, \lambda) = (b, \ldots, b)$, where $b \in \{0, 1\}$ is a uniformly distributed bit. A δ-bias, t-secure coin-flipping protocol is a protocol that (δ, t)-securely computes f^n_{cf} with full security.*

For any $n \in \mathbb{N}$, $t < n$, and $r = r(\kappa)$, Awerbuch et al. [3] presented an r-round, $O(t/\sqrt{r})$-bias coin-flipping protocol tolerating up to t corrupted parties (assuming OWF). Beimel et al. [6] improved this result, giving an r-round, $O(1/\sqrt{r-t})$-bias coin-flipping protocol tolerating up to $t = \beta \cdot n$ corrupted parties, for $0 < \beta < 1$ (assuming OT). We now show how to reduce the dependency on t.

Corollary 3.21. *Assume that TDP and CRH exist. Let $n' < n$ be integers, and let $0 < \beta < \beta' < 1$ be constants. If there exists an n'-party, δ'-bias, r'-round coin-flipping protocol π' tolerating $t' = \beta' n'$ corrupted parties, then there exists an n-party, $(\delta' + \mathsf{err}(n, n', \beta, \beta'))$-bias, $O(t' + r')$-round coin-flipping protocol, tolerating $t = \beta n$ corrupted parties.*

Proof. Note that π' is an n'-party, r'-round protocol that (δ', t')-securely computes $f^{n'}_{\mathsf{cf}}$ with full security. Hence, the proof follows from Item 2 of Theorem 3.1. □

Corollary 3.22 (restating Corollary 1.2)**.** *Assume that TDP and CRH exist. Let $n \in \mathbb{N}$, let $t = \beta n$ for $0 < \beta < 1$, and let $r : \mathbb{N} \mapsto \mathbb{N}$ be an efficiently computable function.*

1. *There exists an n-party, $r(\kappa)$-round, $O\Big(\frac{1}{\sqrt{r(\kappa) - \log(\kappa)}} + \frac{1}{\kappa \cdot \log(\kappa)}\Big)$-bias, t-secure coin-flipping protocol.*
2. *Let $\varphi = \omega(1)$. There exists an n-party, $r(\kappa)$-round, $O\Big(\frac{1}{\sqrt{r(\kappa) - \varphi(\kappa) \cdot \log(\kappa)}}\Big)$-bias, t-secure coin-flipping protocol.*

Proof. For n and β as above, fix $\beta' = (1+\beta)/2$. For Item 1 consider $n' = \log(\kappa)$ and for Item 2 consider $n' = \varphi(\kappa) \cdot \log(\kappa)$.[19] Set $t' = \beta' n'$ and let π' be the protocol of [6] that is an n'-party, r'-round, $O(1/\sqrt{r'-t'})$-round coin-flipping protocol, tolerating up to t' corrupted parties. The proof follows by Corollary 3.21. □

4 Fairness to Full Security for with-Input Functionalities

In this section, we present a reduction from a fully secure computation to a fair computation for functionalities with inputs. The main additional challenge compared to no-input functionalities (Section 3) is enforcing the small committees to carry out the computation on the inputs of the honest parties, while preserving privacy. In Section 4.1, we show how to reduce full security to fairness with restricted identifiable abort in a round-efficient manner. In Section 4.2, we present an analogue result in the honest-majority setting that is unconditional and black-box in the underlying fair protocol. Applications are found in Section 4.3.

Since we consider delegating a computation of functionalities *with inputs* to a small committee, we use a protocol for computing the function over secret-shared inputs. In the honest-majority setting, the parties use a $(t'+1)$-out-of-n' ECSS scheme (Definition 2.2) to distribute their inputs among the committee members. We denote by $\mathsf{SS}_{\mathsf{in\text{-}out}}^{n\to(t',n')}(f)$ the Reconstruct-Compute-Share variant of f, that receives secret shares of the inputs, computes f over the reconstructed n-tuple, and outputs secret shares of the result. In the no-honest-majority setting, every party initially commits to its input (in a somewhat non-trivial way, in order to identify cheating committee members). We denote by $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f)$ the Verify-Reconstruct-Compute variant of f, that receives n'-out-of-n' secret shares for the decommitment of each party, verifies that all the commitments can be opened, reconstructs the n-tuple, computes f, and outputs the result. Formal definitions of these functionalities can be found in Section 4.4. As before, for integers $n' < n$ and for $0 < \beta < \beta' < 1$ we define $\mathsf{err}(n, n', \beta, \beta') = \frac{n}{n'} \cdot e^{-\frac{(\beta'-\beta)^2 n'}{2(1-\beta)}}$.

We prove the following theorem.

Theorem 4.1 (restating Theorem 1.4)**.** *Assume that TDP, CRH, and non-interactive perfectly binding commitment schemes exist. Let f be an n-party functionality with public output, let $0 < \beta < \beta' < 1$, let $n' = \min(n, \log(\kappa) \cdot \varphi(\kappa))$ with $\varphi = 1/\sqrt{1-\beta'} + \Omega(1)$, and let $t = \beta n$ and $t' = \beta' n'$.*

1. *If $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f)$ can be (δ', t')-securely computed with fairness by an r'-round protocol, then f can be $(t' \cdot \delta' + \mathsf{err}(n, n', \beta, \beta'), t)$-securely computed with full security by an $O(t' \cdot r')$ protocol.*

2. *For a deterministic functionality f, if $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f)$ can be $(\delta', n'-1)$-securely computed with fairness by an r'-round protocol, ℓ-times in parallel, for $\ell = \kappa^{\log(e)\cdot\left(\frac{2}{e}+\frac{1}{\varphi(\kappa)}\right)}$, then f can be $(\varphi(\kappa)^2 \cdot \ell \cdot \delta' + \mathsf{err}(n, n', \beta, \beta'), t)$-securely computed with full security by an $O(\varphi(\kappa)^2 \cdot r')$ protocol.*

3. *For $\beta' < 1/2$, the following holds unconditionally. If $\mathsf{SS}_{\mathsf{in\text{-}out}}^{n\to(t',n')}(f)$ can be (δ', t')-securely computed with abort by an r'-round protocol, ℓ-times in parallel, for $\ell = \kappa^{\log(e)\cdot\left(\frac{2}{e}+\frac{1}{\varphi(\kappa)}\right)}$, then f can be $(\varphi(\kappa)^2 \cdot \ell \cdot \delta' + \mathsf{err}(n, n', \beta, \beta'), t)$-securely computed with full security by an r'-round protocol.*

[19] If $n < \varphi(\kappa) \cdot \log(\kappa)$, the parties can simply consider $n' = n$ and $t' = t$.

The proof of Theorem 4.1 is given in the sections below, where the first part follows from a combination of Theorems 4.2 and 4.6; the second part from Theorems 4.4 and 4.6; and the third part from Theorems 4.4 and 4.14.

4.1 Fairness to Full Security without an Honest Majority (with Inputs)

We start by constructing a reduction from full security to fairness for functionalities with inputs, when an honest majority is *not* assumed. In Section 4.1.1, we show how to compute f with full security in the hybrid model computing f with fairness and restricted identifiable abort using $\omega(1)$ calls. In Section 4.1.2, we show how to compile a fair protocol for $\mathsf{SS}^{n\to n'}_{\mathsf{in}}(f)$ to a fair protocol for f with restricted identifiable abort.

4.1.1 Fairness with Restricted Identifiable Abort to Full Security

We now show how to reduce full security to fairness with restricted identifiable abort. Similarly to Section 3.1.1, we start with the simpler case where a single committee $\mathcal{C}$ is considered (i.e., $\ell = 1$). For deterministic functionalities, we can use the technique from Section 3.2.1 and reduce the round complexity by using multiple committees.

Theorem 4.2. *Let f be an n-party, public-output functionality, let $n' < n$, let $0 < \beta < \beta' < 1$, and let $t = \beta n$ and $t' = \beta' n'$. Then, f can be $(\mathsf{err}(n, n', \beta, \beta'), t)$-securely computed with full security in a hybrid model that computes f with fairness and (n', t')-identifiable-abort, by using $t'+1$ sequential calls to the ideal functionality.*

Proof. The protocol is very similar to Protocol 3.3; we present it here for completeness.

Protocol 4.3. *(fairness with restricted identifiable abort to full security)*

- ***Hybrid Model:*** *The protocol is defined in the hybrid model computing f_{elect} with full security, and f with fairness with (n', t')-identifiable-abort.*
- ***Common Input:*** *The values $t', n' \in \mathbb{N}$.*
- ***Private Input:*** *Every party P_i has private input $x_i \in \{0,1\}^*$, for $i \in [n]$.*
- ***The Protocol:***

1. *All the parties invoke $f_{\mathsf{elect}}(n, n', \beta')$ and elect a committee $\mathcal{C}_1 \subset [n]$ of size n'.*
2. *For $i = 1, \ldots, t' + 1$ do*
   (a) *All parties that have not been previously identified send their inputs to the trusted party computing $(f, \mathcal{C}_1\text{-}\mathsf{id}\text{-}\mathsf{fair})$, where the party with the lowest index in $\mathcal{C}_i$ simulates all parties in $\mathcal{C}_1 \setminus \mathcal{C}_i$, using their predetermined default input values. Denote the output P_j receives by y_j.*
   (b) *Every party P_j checks if y_j is a valid output, if so P_j outputs y_j and halts. Otherwise, all parties received $(\perp, i^*)$ as output, where $i^* \in \mathcal{C}_1 \cap \mathcal{I}$. If $i^* \notin \mathcal{C}_i$ (and so P_{i^*} is a previously identified corrupted party), then all parties set i^* to be the party with the lowest index in $\mathcal{C}_i$.*

(c) All parties set $\mathcal{C}_{i+1} = \mathcal{C}_i \setminus \{i^*\}$.

Proving the security of Protocol 4.3 follows almost identically as the proof of Theorem 3.2. The difference is that when simulating the ideal functionality computing f with $\mathcal{C}$-identifiable-abort in a non-aborting iteration, the simulator gets input values from the adversary, which it sends to the trusted party in the fully secure ideal computation in order to get the output. □

4.1.2 Reducing the Round Complexity

We next show that the technique, used in Section 3.2.1 for the honest-majority setting, to reduce the number of rounds for invoking the ideal computation from $\omega(\log(\kappa))$ to $\omega(1)$, can be applied also to the no-honest-majority case. However, this improvement will turn out to be meaningful only for deterministic functionalities (with public output). It is well known that every functionality can be adjusted to be deterministic and with public output via standard techniques; however, the resulting functionality has inputs (even if the original functionality had no inputs); therefore, it is only relevant for the with-input setting.

Theorem 4.4. *Let f be an n-party functionality, let $0 < \beta < \beta' < 1$, let $n' = \log(\kappa)\cdot(\varphi(\kappa) - 1/\varphi(\kappa))$ with $\varphi(\kappa) = 1/\sqrt{1-\beta'} + \Omega(1)$, let $t' = n' - 1$, let $\ell = \kappa^{\log(e)\cdot\left(\frac{2}{e}+\frac{1}{\varphi(\kappa)}\right)}$, and let $t = \beta n$. Then, f can be $(\mathsf{err}(n, n', \beta, \beta'), t)$-securely computed with full security in a hybrid model that computes f with fairness and (ℓ, n', t')-identifiable-abort, by invoking the ideal functionality in $\varphi(\kappa)^2$ rounds.*

Proof sketch. Assume for simplicity of exposition that $n = \Omega(\log(\kappa)\cdot\varphi(\kappa))$. Similarly to the proof of Theorem 3.10, the parties initially invoke $f_{\mathsf{elect}}(n, m, \beta')$ to elect a committee $\mathcal{C} \subseteq [n]$ of size $m = \min(n, \log(\kappa)\cdot\varphi(\kappa))$, and consider all subsets of $\mathcal{C}$ of size $n' = m - n''$ for $n'' = \log(\kappa)/\varphi(\kappa)$, denoted $\boldsymbol{\mathcal{C}} = (\mathcal{C}_1, \ldots, \mathcal{C}_{\ell'})$. Following Claim 3.11, there are polynomially many sub-committees, more precisely, $\ell' \leq \ell$. In addition, following Claim 3.13 the protocol will complete within $\varphi(\kappa)^2$. It is left to show that in every sub-committee there is an honest party. This is done by slightly adjusting Claim 3.12.

Claim 4.5. *In every sub-committee $\mathcal{C}_i$, there exists at least one honest party.*

Proof. We require that $n' > |\mathcal{C} \cap \mathcal{I}|$. By the definition of $f_{\mathsf{elect}}(n, m, \beta')$ it holds that $|\mathcal{C} \cap \mathcal{I}| \leq \beta' \cdot m$. The claim will therefore follow if $n' > \beta' \cdot m$, i.e.,

$$\log(\kappa)\cdot(\varphi(\kappa) - 1/\varphi(\kappa)) > \beta' \cdot \log(\kappa)\cdot\varphi(\kappa). \tag{2}$$

Since $0 < \beta' < 1$ and $\varphi = \Omega(1)$, Equation (2) holds for

$$\varphi(\kappa) > \frac{1}{\sqrt{1-\beta'}}.$$

□

The simulator proceeds similarly to the simulator in Theorem 3.2, where in every iteration, if the adversary sends $(\mathsf{abort}, \mathcal{J})$, with $\mathcal{J} \subseteq \mathcal{I}$ and $\mathcal{J} \cap \mathcal{C}_l \neq \emptyset$ for every $l \in [\ell']$, the simulator S simulates sending $(\perp, \mathcal{J})$ to the parties. If A sends input values $\{x'_i\}_{i\in\mathcal{I}}$ to the computation, S forwards these input values to the trusted party and obtains the output y. Next, S forwards y to the adversary and outputs whatever A outputs. □

4.1.3 Fairness to Fairness with Restricted Identifiable Abort

In this section, we present a reduction from fair computation with identifiable abort to fair computation for functionalities with inputs. More specifically, let π' be a protocol for computing $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f)$ with fairness, even when run ℓ times in parallel by subsets of parties $\boldsymbol{\mathcal{C}} = (\mathcal{C}_1, \ldots, \mathcal{C}_\ell)$. We show that π' can be compiled into a protocol π that computes f with fairness and $\boldsymbol{\mathcal{C}}$-identifiable-abort. For $\ell > 1$ we require that f is deterministic.

The above is formally stated in the theorem below, using the following notations. Let f be a deterministic n-party functionality with public output, let $t, n' < n$, let $t' < n'$ and $\ell \in \mathsf{poly}(n)$, and let $\boldsymbol{\mathcal{C}} = (\mathcal{C}_1, \ldots, \mathcal{C}_\ell)$, where $\mathcal{C}_1, \ldots, \mathcal{C}_\ell \subseteq [n]$ of size n'.

Theorem 4.6. *Assume that TDP, CRH, and non-interactive perfectly binding commitment schemes exist. Then, there exists a* PPT *algorithm* $\mathsf{Compiler}^{n'\to n}$ *such that for any n'-party, r'-round protocol π' computing $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f)$, the protocol $\pi = \mathsf{Compiler}^{n'\to n}(\pi', \boldsymbol{\mathcal{C}})$ is an n-party, $O(r')$-round protocol computing f with the following guarantee. If the number of corrupted parties in every $\mathcal{C}_j$ is at most t', and π' is a protocol that (δ', t')-securely computes $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f)$ with fairness, ℓ times in parallel, then π is a protocol that $(\ell\cdot\delta', t)$-securely computes f with fairness and $\boldsymbol{\mathcal{C}}$-identifiable-abort.*

Proof. In Lemma 4.7, we construct the protocol compiler $\mathsf{Compiler}^{n\to n'}(\pi', \boldsymbol{\mathcal{C}})$ in the $(f_{\mathsf{aug\text{-}ct}}, \mathsf{ZK}^{1:\mathrm{M}})$-hybrid model (explained below), and in Lemma 2.15, we show how to instantiate the ideal functionalities $f_{\mathsf{aug\text{-}ct}}$ and $\mathsf{ZK}^{1:\mathrm{M}}$ in a round-preserving manner. The proof follows from the sequential composition theorem (Proposition 2.7). □

A high-level description of the compiler. We now describe the compiler at a high level. For simplicity, we consider a single committee $\mathcal{C}$ that computes $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f)$ with fairness and identifiable abort. The compiler consists of three phases: initially, every party shares its input among the committee members; next, the committee computes f over the shared inputs; and finally, the committee members distribute the output to all the parties.

In the first phase, it is important that the input values of the parties are shared in a way that forces each committee member to use the actual value it received from each party (otherwise, corrupted parties might learn the value of f on inputs that are correlated to the actual input values of the honest parties). In case a corrupted committee member uses a different value, it should be identified *without* learning the output. Note that the guarantee provided by a fair computation is that if the adversary learns the output then so do all honest parties, but there is no restriction on the adversary from entering arbitrary values to the computation. One way to solve this issue is by having the functionality $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f)$ itself identify committee members that modify their shares to all the parties *before* computing the function f. However, now it should be ensured that a corrupted party cannot falsely incriminate an honest committee member, and claim that the committee member used different values than the one it gave him.

We solve both issues by having each party first publicly commit the values it sends to the committee members, in the following way. Every party secret shares its input value and publicly commits to each share. Next, the party encrypts each share using a committing public-key encryption scheme[20] (every share is encrypted with the public key of the recipient committee member), and broadcasts all ciphertexts. Finally, each party proves to all other parties that it behaved honestly.

[20] A committing public-key encryption scheme is an encryption scheme with the property that it is computationally infeasible to find two pairs of (different) plaintext and randomness that are encrypted to the same ciphertext.

In the second phase, the committee members compute $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f)$ over the broadcast channel. This computation is compiled to enforce a semi-honest behavior in a similar way to the GMW compiler [33]. The committee members run an augmented coin-tossing protocol, where each committee member receives a random string, and all other parties receive a commitment to this string. Next, the committee members execute the fair protocol computing $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f)$ with identifiable abort until it completes with an output value y or with an identity of a corrupted party in the committee.

In the third phase, each committee member broadcasts its output value and proves to all parties that it behaved honestly, i.e., that it followed the protocol using as input the shares it decrypted from the ciphertexts in the first phase, and the committed randomness generated in the second phase. If a party receives an accepting proof, it outputs the corresponding output value. Assuming that there exists an honest party in the committee, at least one of the proofs will be accepting.

If during the first two phases a party in the committee is identified as corrupted, then every party outputs its identity and the protocol halts. If a party outside of the committee is identified, then all the parties remove this party from the computation and resume the execution.

Ideal functionalities used in the compiler. We construct the compiler in the hybrid model computing $f_{\mathsf{aug\text{-}ct}}^{\mathcal{C}}$ and $\mathsf{ZK}^{1:\mathrm{M}}$ (see Definitions 2.13 and 2.14). In the compiler below we consider two types of NP relations. The first is used when a party sends its shared input to the committees, and the second when a committee member sends its output value to all the parties. These relation are formally described as follows:

- In the relation R_{enc}, parametrized by a public-key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and a non-interactive commitment scheme Com, the public statement $(\boldsymbol{ek}, \boldsymbol{e}, \boldsymbol{c})$ consists of a vector of encryption keys $\boldsymbol{ek} = (ek_{1,1}, \ldots, ek_{n',\ell})$, a vector of ciphertexts $\boldsymbol{e} = (e_{1,1}, \ldots, e_{n',\ell})$ and a vector of commitments $\boldsymbol{c} = (c_{1,1}, \ldots, c_{n',\ell})$; the witness $(\hat{\boldsymbol{s}}, \boldsymbol{r})$ consists of vectors of strings $\hat{\boldsymbol{s}} = (\hat{s}_{1,1}, \ldots, \hat{s}_{n',\ell})$, where $\hat{s}_{j,l} = (s_{j,l}, \rho_{j,l})$, and randomness $\boldsymbol{r} = (r_{1,1}, \ldots, r_{n',\ell})$. For every $j \in [n']$ and $l \in [\ell]$, it holds that $e_{j,l} = \mathsf{Enc}_{ek_{j,l}}(\hat{s}_{j,l}; r_{j,l})$, that $c_{j,l} = \mathsf{Com}(s_{j,l}; \rho_{j,l})$, and in addition, by denoting $x_l = \bigoplus_{j\in[n']} s_{j,l}$, that $x_1 = \ldots = x_\ell$.
- In the following vector of NP-relations $(R_1, \ldots, R_{n'})$, parametrized by a public-key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and a commitment scheme Com, for every $j \in [n']$, the relation R_j contains pairs $((\boldsymbol{e}, \boldsymbol{m}, \sigma, \mathsf{out}), (dk, r, \rho))$, where the public instance consists of a vector of ciphertexts $\boldsymbol{e} = (e_1, \ldots, e_n)$, a vector of messages $\boldsymbol{m} = (m_1, \ldots, m_p)$, a commitment σ, and a value out. The witness consists of a decryption key dk, a random string r, and a decommitment information ρ. It holds that out is the output value under the next-message function of P_j in protocol π' on input $(\hat{s}_1, \ldots, \hat{s}_n) = (\mathsf{Dec}_{dk}(e_1), \ldots, \mathsf{Dec}_{dk}(e_n))$, randomness r and messages $\boldsymbol{m}$; in addition, $\sigma = \mathsf{Com}(r; \rho)$.

Lemma 4.7. *Assume that committing public-key encryption schemes and non-interactive perfectly binding commitment schemes exist, and consider the same notations as in Theorem 4.6. If π' is a (δ', t')-secure protocol computing $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f)$ with fairness, ℓ times in parallel, then the protocol $\pi = \mathsf{Compiler}^{n'\to n}(\pi', \boldsymbol{\mathcal{C}})$ is an n-party protocol that $(\ell \cdot \delta', t)$-securely computes f with fairness and $\boldsymbol{\mathcal{C}}$-identifiable-abort, in the $(f_{\mathsf{aug\text{-}ct}}, \mathsf{ZK}^{1:M})$-hybrid model.*

Proof. The proof proceeds in a similar way to the proof of Theorem 3.4, where the main difference relates to sending the shared inputs to each committee. As in Theorem 3.4, we assume without loss of generality that π' is fair with identifiable abort and that all messages are sent over the broadcast channel. The n-party protocol $\pi = \mathsf{Compiler}^{n'\to n}(\pi', \boldsymbol{\mathcal{C}})$ is defined as follows.

Construction 4.8. *(fairness to fairness with restricted identifiable abort)*

- ***Hybrid Model:*** *The protocol is defined in the hybrid model computing $f_{\mathsf{aug\text{-}ct}}$ with restricted identifiable abort and $\mathsf{ZK}^{1:M}$ with full security.*
- ***Common Input:*** *A public-key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, a non-interactive commitment scheme Com, an n'-party protocol π', and $\boldsymbol{\mathcal{C}} = (\mathcal{C}_1, \ldots, \mathcal{C}_\ell)$. We use the notation P_j^l to refer to the j'th party in $\mathcal{C}_l$.*
- ***Private Input:*** *Every party P_i, for $i \in [n]$, has private input $x_i \in \{0,1\}^*$.*
- ***The Protocol:*** *Let $\mathcal{J}$ denote the (initially empty) set of identified corrupted parties. At any point in the protocol, we call a committee $\mathcal{C}_l$* active *if $\mathcal{C}_l \cap \mathcal{J} = \emptyset$.*

1. *Every $\mathsf{P}_j \in \bigcup_{l\in[\ell]} \mathcal{C}_l$ generates $(dk_j, ek_j) \leftarrow \mathsf{Gen}(1^\kappa)$ and broadcasts ek_j. If some $\mathsf{P}_{i^*} \in \bigcup \mathcal{C}_l$ did not broadcast, every party P_i adds i^* to $\mathcal{J}$. (The key-pair generated by $\mathsf{P}_j^l \in \mathcal{C}_l$ is denoted by $(dk_{j,l}, ek_{j,l})$.) Denote $\boldsymbol{ek} = (ek_{1,1}, \ldots, ek_{n',\ell})$.*
2. *Every party P_i proceeds as follows:*
   (a) *For every active $\mathcal{C}_l$, P_i samples random values $(s_i^{1,l}, \ldots, s_i^{n',l})$, conditioned on $x_i = \bigoplus_j s_i^{j,l}$. For every $j \in [n']$, P_i commits to the j'th share as $c_i^{j,l} = \mathsf{Com}(s_i^{j,l}; \rho_i^{j,l})$, denote $\hat{s}_i^{j,l} = (s_i^{j,l}, \rho_i^{j,l})$, and encrypt $e_i^{j,l} = \mathsf{Enc}_{ek_{j,l}}(\hat{s}_i^{j,l}; r_i^{j,l})$.*
   (b) *Send the values $((\boldsymbol{ek}, \boldsymbol{e}_i, \boldsymbol{c}_i), (\hat{\boldsymbol{s}}_i, \boldsymbol{r}_i))$ to $\mathsf{ZK}^{1:M}$ (parametrized by R_{enc}),[21] where $\boldsymbol{e}_i = (e_i^{1,1}, \ldots, e_i^{n',\ell})$, $\boldsymbol{c}_i = (c_i^{1,1}, \ldots, c_i^{n',\ell})$, $\hat{\boldsymbol{s}}_i = (\hat{s}_i^{1,1}, \ldots, \hat{s}_i^{n',\ell})$, and $\boldsymbol{r}_i = (r_i^{1,1}, \ldots, r_i^{n',\ell})$.*
   If $\mathsf{ZK}^{1:M}$ returned $((\boldsymbol{ek}, \boldsymbol{e}_{i^*}, \boldsymbol{c}_{i^*}), 0)$ for some P_{i^*}, then all parties hard-wire to f the default value for P_{i^*} and remove P_{i^*} from the party-set; if $i^* \in \bigcup \mathcal{C}_l$, then every P_i adds i^* to $\mathcal{J}$.
3. *For every active $\mathcal{C}_l$, every party $\mathsf{P}_j^l \in \mathcal{C}_l$ decrypts the vector $\boldsymbol{e}_{j,l} = (e_1^{j,l}, \ldots, e_n^{j,l})$, by computing $\hat{s}_i^{j,l} = \mathsf{Dec}_{dk_{j,l}}(e_i^{j,l})$, for every $i \in [n]$. In addition, parse $\hat{s}_i^{j,l} = (s_i^{j,l}, \rho_i^{j,l})$.*
4. *For every active $\mathcal{C}_l$, all parties invoke $f_{\mathsf{aug\text{-}ct}}^{\mathcal{C}_l}$ with $\mathcal{C}_l$-identifiable-abort (in parallel);[22] every $\mathsf{P}_j^l \in \mathcal{C}_l$ receives back $(r_{j,l}, \rho_{j,l}, \boldsymbol{\sigma}_l)$ where $\boldsymbol{\sigma}_l = (\sigma_{1,l}, \ldots, \sigma_{n',l})$ is a public output. In case the computation for $\mathcal{C}_l$ aborts and some party $\mathsf{P}_{i^*} = \mathsf{P}_{j^*}^l \in \mathcal{C}_l$ is identified as corrupted, every P_i adds i^* to $\mathcal{J}$.*
5. *For every active $\mathcal{C}_l$, the parties in $\mathcal{C}_l$ execute the protocol π' for computing $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f)$, parametrized by $\boldsymbol{c} = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n)$, over the broadcast channel, where $\mathsf{P}_j^l \in \mathcal{C}_l$ uses $(\hat{s}_1^{j,l}, \ldots, \hat{s}_n^{j,l})$ as its input and $r_{j,l}$ as its random coins. Denote by $\boldsymbol{m}_{j,l} = (m_1^{j,l}, \ldots, m_p^{j,l})$ the messages $\mathsf{P}_j^l \in \mathcal{C}_l$ received during the protocol and let $\mathsf{out}_{j,l}$ be the output $\mathsf{P}_j^l \in \mathcal{C}_l$ received (either a valid value y or $(\perp, i^*)$ with $i^* \in \mathcal{C}_l$).*

[21] More formally, we consider an ideal world for n-bounded parallel computation of $\mathsf{ZK}^{1:M}$, where P_i acts as the prover in the i'th computation.

[22] More formally, the parties invoke the functionality computing at once ℓ instances of $f_{\mathsf{aug\text{-}ct}}$, where the l'th instance has $\mathcal{C}_l$-identifiable-abort.

6. *For every active $\mathcal{C}_l$, every $\mathsf{P}_j^l \in \mathcal{C}_l$ proves to all parties that $\mathsf{out}_{j,l}$ is indeed its output value, i.e., P_j^l sends $((\boldsymbol{e}_{j,l}, \boldsymbol{m}_{j,l}, \sigma_{j,l}, \mathsf{out}_{j,l}), (dk_j, r_{j,l}, \rho_{j,l}))$ to $\mathsf{ZK}^{1:M}$ (parametrized by R_j).*[23] *Once party P_i receives $((\boldsymbol{e}_{j,l}, \boldsymbol{m}_{j,l}, \sigma_{j,l}, \mathsf{out}_{j,l}), 1)$ where the values $(\boldsymbol{e}_{j,l}, \boldsymbol{m}_{j,l}, \sigma_{j,l})$ match the common view, and $\mathsf{out}_{j,l} = (\perp, i^*)$ (for some $\mathsf{P}_{i^*} = \mathsf{P}_{j^*}^l \in \mathcal{C}_l$), P_i adds i^* to $\mathcal{J}$; otherwise, P_i adds $\mathsf{out}_{j,l}$ to the (initially empty) set $\mathcal{V}_i$. If P_i receives an invalid proof from some party $\mathsf{P}_{i^*} = \mathsf{P}_{j^*}^l \in \mathcal{C}_l$, P_i adds i^* to $\mathcal{J}$.*

7. *If $\mathcal{V}_i \neq \emptyset$, party P_i arbitrarily chooses $y \in \mathcal{V}_i$ and outputs y. Otherwise, P_i outputs $(\perp, \mathcal{J})$.*

Let A be an adversary attacking the execution of protocol π in the $(f_{\mathsf{aug\text{-}ct}}, \mathsf{ZK}^{1:\mathrm{M}})$-hybrid model and let $\mathcal{I} \subseteq [n]$ be a subset of size at most t, satisfying $|\mathcal{I} \cap \mathcal{C}_l| \leq t'$, for every $l \in [\ell]$. We construct the following adversary S for the ideal model computing f with fairness and $\boldsymbol{\mathcal{C}}$-identifiable-abort. On inputs $\{x_i\}_{i\in\mathcal{I}}$ and auxiliary input z, the simulator S starts by emulating A on these inputs. S plays toward A the roles of the honest parties and the ideal functionalities $f_{\mathsf{aug\text{-}ct}}$ and $\mathsf{ZK}^{1:\mathrm{M}}$. S initializes empty sets $\mathcal{J}$ and $\mathcal{V}$. For simplicity, assume that all input values and random strings are elements in $\{0,1\}^\kappa$.

To simulate Step 1, the simulator S sends to A public keys for honest parties in $\bigcup \mathcal{C}_l$ and receives from A public keys for corrupted parties in $\bigcup \mathcal{C}_l$; if A does not provide a public key for $\mathsf{P}_{i^*} = \mathsf{P}_{j^*}^l \in \bigcup \mathcal{C}_l$, the simulator S adds i^* to $\mathcal{J}$. Denote by $ek_{j,l}$ the public key for $\mathsf{P}_j^l \in \mathcal{C}_l$ and let $\boldsymbol{ek} = (ek_{1,1}, \ldots, ek_{n',\ell})$.

To simulate Step 2, the simulator S proceeds as follows for every honest party P_i. In Step 2a, for every active $\mathcal{C}_l$ (satisfying $\mathcal{C}_l \cap \mathcal{J} = \emptyset$), compute secret shares of zero, $0^{2\kappa} = \bigoplus_{j\in[n']} s_i^{j,l}$, commit to each share as $c_i^{j,l} = \mathsf{Com}(s_i^{j,l}; \rho_i^{j,l})$, denote $\hat{s}_i^{j,l} = (s_i^{j,l}, \rho_i^{j,l})$, and encrypt $e_i^{i,j} \leftarrow \mathsf{Enc}_{ek_{j,l}}(\hat{s}_i^{j,l})$. In Step 2b, for every active $\mathcal{C}_l$, denote $\boldsymbol{e}_i = (e_i^{1,1}, \ldots, e_i^{n',\ell})$, and $\boldsymbol{c}_i = (c_i^{1,1}, \ldots, c_i^{n',\ell})$. Next, S simulates $\mathsf{ZK}^{1:\mathrm{M}}$ as follows. On behalf of every honest party P_i, the simulator S sends $((\boldsymbol{ek}, \boldsymbol{e}_i, \boldsymbol{c}_i), 1)$ to A; on behalf of every corrupted party $\tilde{\mathsf{P}}_i$, the simulator S receives values $((\boldsymbol{ek}, \boldsymbol{e}_i, \boldsymbol{c}_i), (\hat{\boldsymbol{s}}_i, \boldsymbol{r}_i))$ from A, verifies the relation R_{enc} and answers A accordingly.

To simulate Step 4, for every active $\mathcal{C}_l$, the simulator S emulates $f_{\mathsf{aug\text{-}ct}}^{\mathcal{C}_l}$ by sampling random strings $(r_{j,l}, \rho_{j,l})$ for parties $\mathsf{P}_j^l \in \mathcal{C}_l$, computing $\sigma_{j,l} = \mathsf{Com}(r_{j,l}; \rho_{j,l})$ and setting $\boldsymbol{\sigma}_l = (\sigma_{1,l}, \ldots, \sigma_{n',l})$. Next, S hands $(r_{j,l}, \rho_{j,l}, \boldsymbol{\sigma}_l)$ to corrupted parties $\tilde{\mathsf{P}}_j \in \mathcal{C}_l$ and $(\lambda, \boldsymbol{\sigma}_l)$ to corrupted parties outside of $\mathcal{C}_l$. In case S receives (abort, i^*) with $i^* \in \mathcal{I} \cap \mathcal{C}_l$ from A, it responds with $(\perp, i^*)$ and adds i^* to $\mathcal{J}$.

Next, the simulator S uses the simulator $\tilde{\mathsf{S}}$ that is guaranteed to exist for the ℓ-times parallel execution of π' when interacting with the residual adversary of A in Step 5. The simulator S invokes $\tilde{\mathsf{S}}$ on input values $\{\hat{\boldsymbol{s}}_{j,l}\}_{j\in\mathcal{C}_l\cap\mathcal{I}}$ for the l'th execution of π' with parties in $\mathcal{C}_l$ (where the values $\hat{\boldsymbol{s}}_{j,l} = (\hat{s}_1^{j,l}, \ldots, \hat{s}_n^{j,l})$ are obtained during the simulation of Step 2b) and on auxiliary information containing z, the input values $\{x_i\}_{i\in\mathcal{I}}$, and the transcript of the simulation until this point. If $\tilde{\mathsf{S}}$ sends (abort, i^*) for the l'th computation, with $i^* \in \mathcal{I} \cap \mathcal{C}_l$, the simulator S adds i^* to $\mathcal{J}$. Otherwise, $\tilde{\mathsf{S}}$ sends input values $\{\tilde{\boldsymbol{s}}_{j,l}\}_{j\in\mathcal{C}_l\cap\mathcal{I}}$ with $\tilde{\boldsymbol{s}}_{j,l} = (\tilde{s}_1^{j,l}, \ldots, \tilde{s}_n^{j,l})$. The simulator S verifies that $\tilde{s}_i^{j,l} = (s_i^{j,l}, \rho_i^{j,l})$ and that $c_i = \mathsf{Com}(s_i^{j,l}; \rho_i^{j,l})$; if the verifications fails, S responds to $\tilde{\mathsf{S}}$ with $(\perp, j^*)$ (for the smallest such j^*) and adds i^* to $\mathcal{J}$, such that $\mathsf{P}_{i^*} = \mathsf{P}_{j^*}^l \in \mathcal{C}_l$. In case no computation has valid inputs from A, the simulator S sends the set $\mathcal{J}$ to its trusted party and receives back $(\perp, \mathcal{J})$. Otherwise, S reconstructs the input values $\{x_i'\}_{i\in\mathcal{I}}$, by computing $x_i' = \bigoplus_j s_i^{j,l}$ for an arbitrary

[23] More formally, we consider an ideal world for $(n' \cdot \ell)$-bounded parallel computation of $\mathsf{ZK}^{1:\mathrm{M}}$, where every $\mathsf{P}_j^l \in \mathcal{C}_l$ acts as the prover in a separate computation.

computation l with valid inputs. Next, S sends $\{x'_i\}_{i\in\mathcal{I}}$ to the trusted party, receives output y and forwards y to $\tilde{\mathsf{S}}$ for every computation with valid inputs. In both cases, S receives the output from $\tilde{\mathsf{S}}$, which contains the view of the adversary, and interacts with A accordingly.

To simulate Step 6, the simulator S simulates $\mathsf{ZK}^{1:\mathrm{M}}$. The simulator S sends, on behalf of every honest party P_j^l in an active $\mathcal{C}_l$, the message $((\boldsymbol{e}_{j,l}, \boldsymbol{m}_{j,l}, \sigma_{j,l}, \mathsf{out}_{j,l}), 1)$ to every corrupted party, where $\boldsymbol{m}_{j,l}$ is obtained from the output of $\tilde{\mathsf{S}}$ and $\mathsf{out}_{j,l}$ is either the output value y or $(\perp, i^*)$ where $\mathsf{P}_{i^*} = \mathsf{P}_{j^*}^l \in \mathcal{C}_l$ is the identified corrupted party. In addition, S receives $((\boldsymbol{e}_{j,l}, \boldsymbol{m}_{j,l}, \sigma_{j,l}, \mathsf{out}_{j,l}), (dk_j, r_{j,l}, \rho_{j,l}))$ from A on behalf of every corrupted party P_j in $\mathcal{C}_l$ and verifies the relation according to R_j. Finally, S outputs whatever A outputs and halts.

Proving indistinguishability. We prove computational indistinguishability between the real execution of the compiled protocol π running with adversary A and the ideal computation of f running with S via a series of hybrids experiments. The output of each experiment is the output of the honest parties and of the adversary.

The game $\mathrm{HYB}^1_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)$**.** This game is defined to be the execution of the protocol π in the $(f_{\mathsf{aug\text{-}ct}}, \mathsf{ZK}^{1:\mathrm{M}})$-hybrid model on inputs $\boldsymbol{x} \in (\{0,1\}^*)^n$ and security parameter κ with adversary A running on auxiliary information z and controlling parties in $\mathcal{I}$.

The game $\mathrm{HYB}^2_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)$**.** In this game, we modify $\mathrm{HYB}^1_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)$ as follows. Whenever an honest party invokes $\mathsf{ZK}^{1:\mathrm{M}}$ with (x,w) (in Steps 2b and 6), all parties receive output $(x,1)$ without checking if w is a witness for x.

Claim 4.9. $\{\mathrm{HYB}^1_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)\}_{\boldsymbol{x},z,\kappa} \approx \{\mathrm{HYB}^2_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)\}_{\boldsymbol{x},z,\kappa}$.

Proof. This is immediate since honest parties always send valid witnesses to $\mathsf{ZK}^{1:\mathrm{M}}$. □

The game $\mathrm{HYB}^3_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)$**.** In this game, we modify $\mathrm{HYB}^2_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)$ as follows. In Step 5, instead of running ℓ instances of protocol π' in parallel, use the simulator $\tilde{\mathsf{S}}$ that is guaranteed to exists for the residual adversary A.

More specifically, $\tilde{\mathsf{S}}$ is invoked on input values $\{\boldsymbol{s}_{j,l}\}_{j\in\mathcal{C}_l\cap\mathcal{I}}$ for the l'th execution of π' with parties in $\mathcal{C}_l$ (where $\boldsymbol{s}_{j,l} = (s_1^{j,l}, \ldots, s_n^{j,l})$ are the values sent by A to $\mathsf{ZK}^{1:\mathrm{M}}$ in Step 2) and on auxiliary information containing z, the input values $\{x_i\}_{i\in\mathcal{I}}$ and the transcript of the experiment until this point. If $\tilde{\mathsf{S}}$ sends (abort, i^*) for the l'th computation, with $i^* \in \mathcal{I}\cap\mathcal{C}_l$, add i^* to $\mathcal{J}$. Otherwise, $\tilde{\mathsf{S}}$ sends input values $\{\tilde{\boldsymbol{s}}_{j,l}\}_{j\in\mathcal{C}_l\cap\mathcal{I}}$ with $\tilde{\boldsymbol{s}}_{j,l} = (\tilde{s}_1^{j,l}, \ldots, \tilde{s}_n^{j,l})$; If for some $j \in \mathcal{C}_l \cap \mathcal{I}$ the signed values are not verified, reply with $(\perp, j)$ to $\tilde{\mathsf{S}}$ and add j to $\mathcal{J}$. If there exist instances with verified inputs, reconstruct the input values $\{x'_i\}_{i\in\mathcal{I}}$, where $x'_i = \mathsf{Recon}(\tilde{s}_i^{1,l}, \ldots, \tilde{s}_i^{n',l})$ for an arbitrary computation l with valid inputs (where for $j \notin \mathcal{C}_l \cap \mathcal{I}$, set $\tilde{s}_i^{j,l} = s_i^{j,l}$), and compute f over these inputs and the honest parties' inputs to obtain the output y and forward y to $\tilde{\mathsf{S}}$ for every computation with valid inputs. Next, use the output from $\tilde{\mathsf{S}}$, which contains the view of the adversary, in order to interacts with A.

Claim 4.10. $\{\mathrm{HYB}^2_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)\}_{\boldsymbol{x},z,\kappa} \equiv_c \{\mathrm{HYB}^3_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)\}_{\boldsymbol{x},z,\kappa}$.

Proof. This follows from the security of the simulator $\tilde{\mathsf{S}}$. Indeed, the ability to distinguish between HYB^2 and HYB^3 with non-negligible advantage implies the same advantage in distinguishing between the simulator $\tilde{\mathsf{S}}$ and the execution of the ℓ-times parallel execution of π' when interacting with the residual adversary of A in Step 5. □

The games $\text{HYB}^{4,i^*,j^*,l^*}_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)$, for $1 \leq i^* \leq n$, $1 \leq j^* \leq n'$ and $1 \leq l^* \leq \ell$. In these games, we modify $\text{HYB}^3_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)$ as follows. For every honest party P_i, in addition to generating $(s_i^{1,l},\ldots,s_i^{n',l}) \leftarrow \mathsf{Share}(x_i)$, generate secret shares of zero as $(\tilde{s}_i^{1,l},\ldots,\tilde{s}_i^{n',l}) \leftarrow \mathsf{Share}(0)$. Next, for $(i,j,l) < (i^*,j^*,l^*)$ (i.e., for $i < i^*$, or $i = i^*$ and $j < j^*$, or $i = i^*$, $j = j^*$ and $l < l^*$), compute $c_i^{j,l} \leftarrow \mathsf{Enc}_{pk_j}(\tilde{s}_i^{j,l})$, and for $(i,j,l) \geq (i^*,j^*,l^*)$, compute $c_i^{j,l} \leftarrow \mathsf{Enc}_{pk_j}(s_i^{j,l})$.

Claim 4.11. $\{\text{HYB}^3_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)\}_{\boldsymbol{x},z,\kappa} \equiv_c \{\text{HYB}^{4,n.n',\ell}_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)\}_{\boldsymbol{x},z,\kappa}$.

Proof. Note that $\text{HYB}^3_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa) \approx \text{HYB}^{4,1,1,1}_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)$. For every $i \in [n]$, $j \in [n']$ and $l \in [\ell-1]$, it holds that $\text{HYB}^{4,i,j,l}_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa) \equiv_c \text{HYB}^{4,i,j,l+1}_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)$; similarly, for every $i \in [n]$, $j \in [n'-1]$ it holds that $\text{HYB}^{4,i,j,\ell}_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa) \equiv_c \text{HYB}^{4,i,j+1,1}_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)$ and for every $i \in [n-1]$ it holds that $\text{HYB}^{4,i,n',\ell}_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa) \equiv_c \text{HYB}^{4,i+1,1,1}_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)$. Otherwise, since the simulated computation of π' is independent of the honest parties' inputs, there exists an attack on the semantic security of the encryption scheme. The claim follows using a standard hybrid argument. □

The game $\text{HYB}^5_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)$. In this game, we modify $\text{HYB}^{4,n.n',\ell}_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)$ as follows. Instead of computing f on $\{x_i'\}_{i\in\mathcal{I}}$ and on the input values of the honest parties, send $\{x_i'\}_{i\in\mathcal{I}}$ (or the set $\mathcal{J}$ in case all computations are aborted) to the ideal functionality computing f in the fair ideal model with $(\mathcal{C}_1,\ldots,\mathcal{C}_\ell)$-identifiable abort and get back the output y.

Claim 4.12. $\{\text{HYB}^{4,n.n',\ell}_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)\}_{\boldsymbol{x},z,\kappa} \approx \{\text{HYB}^5_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)\}_{\boldsymbol{x},z,\kappa}$.

Proof. This is immediate since the ideal model computes f on the honest parties' inputs as required. □

The proof of Lemma 4.7 now follows since $\text{HYB}^5_{\pi,\mathcal{I},\mathsf{A}(z)}(\boldsymbol{x},\kappa)$ exactly describes the simulation done by S, and in particular, does not depend on the input values of honest parties. □

Remark 4.13. *The result in Theorem 4.6, of computing an n-party functionality f with fairness and restricted identifiable abort, is achieved by having small committees compute $\mathsf{SS}^{n\to n'}_{\mathsf{in}}(f)$, over committed values, with fairness and identifiable abort. This technique can be extended in a straightforward way to construct a secure computation of f with restricted identifiable abort, by having each of the committees securely compute $\mathsf{SS}^{n\to n'}_{\mathsf{in}}(f)$ without fairness, only with identifiable abort (where the adversary can abort the computation in each committee after learning the output). Moreover, the technique can be extended to a computation with restricted identifiable abort of* single-input reactive functionalities *(see Section 2.3.4) that receive inputs from the parties only at the first call, i.e., the output of all proceeding calls is determined by the input for the first call. Indeed, this translates to having the committees compute $\mathsf{SS}^{n\to n'}_{\mathsf{in}}(f_i)$ (where f_i is the i'th function of the single-input reactive functionality) over the same committed values that were provided by all the parties, and prove in zero knowledge that the outcome of every computation is correct. Looking ahead, this observation will turn out useful in Section 4.3.1.*

4.2 Fairness to Full Security with an Honest Majority (with Inputs)

4.2.1 Fairness with Restricted Identifiable Abort with an Honest Majority

In this section, we show that in the case that an honest majority is guaranteed, the reduction from fairness with restricted identifiable abort to fairness can be much more elegant, and more importantly, be based on much simpler tools. Specifically, we devise a compiler, similar to the one for the no-honest-majority case that is based solely on error-correcting secret-sharing schemes (ECSS), which exist unconditionally (see Definition 2.2).

Theorem 4.14. *Let f, n', ℓ, and $\boldsymbol{\mathcal{C}}$ as in Theorem 4.6, in addition, let $t < n/2$ and $t' < n'/2$. There exists a* PPT *algorithm $\mathsf{Compiler}_{\mathsf{hm}}^{n' \to n}$ such that for any n'-party, r'-round protocol π' computing $\mathsf{SS}_{\mathsf{in\text{-}out}}^{n \to (t',n')}(f)$, the protocol $\pi = \mathsf{Compiler}_{\mathsf{hm}}^{n' \to n}(\pi', \boldsymbol{\mathcal{C}})$ is an n-party, $O(t' \cdot r')$-round protocol computing f with the following guarantee, in the information-theoretic (statistical) setting.*

If the number of corrupted parties in every $\mathcal{C}_j$ is smaller than t', and π' is a protocol that (δ', t')-securely computes $\mathsf{SS}_{\mathsf{in\text{-}out}}^{n \to (t',n')}(f)$ with abort, ℓ times in parallel, then π is a protocol that $(\ell \cdot \delta', t)$-securely computes f with fairness and $\boldsymbol{\mathcal{C}}$-identifiable-abort. Furthermore, the compiler is black-box with respect to the protocol π'.

Proof. Initially, following Lemma 3.17 we can assume that π' is a protocol that (δ', t')-securely computes $\mathsf{SS}_{\mathsf{in\text{-}out}}^{n \to (t',n')}(f)$ with identifiable abort. Next, we construct $\mathsf{Compiler}_{\mathsf{hm}}^{n' \to n}$ by adjusting the compiler for the no-honest-majority setting (Construction 4.8) as follows.

The compiler is initially defined in the Setup-Commit-then-Proof hybrid model (defined in [45]). Upon first invocation of the trusted party (the setup-commit phase), every party P_i obtains correlated randomness $\boldsymbol{r}_i = (r^i_{\mathsf{input}}, \boldsymbol{r}^i_{\mathsf{mask}}, \boldsymbol{r}^i_{\mathsf{prot}})$, where $\boldsymbol{r}^i_{\mathsf{mask}} = (r^{i,1}_{\mathsf{mask}}, \ldots, r^{i,n'}_{\mathsf{mask}})$ and $\boldsymbol{r}^i_{\mathsf{prot}} = (r^{i,1}_{\mathsf{prot}}, \ldots, r^{i,\ell}_{\mathsf{prot}})$. The committed correlated randomness is used as follows:

- r^i_{input} is used to commit to the input of P_i by broadcasting $x_i \oplus r^i_{\mathsf{input}}$.
- $\boldsymbol{r}^i_{\mathsf{mask}}$ is used to mask the communication (over the broadcast channel) between P_i and P_j. These are pairwise correlated values, i.e., $r^{i,j}_{\mathsf{mask}} = r^{j,i}_{\mathsf{mask}}$.
- $\boldsymbol{r}^i_{\mathsf{prot}}$ consists of randomness for executing the protocol π'. If $\mathsf{P}_i \in \mathcal{C}_l$, then P_i will use $r^{i,l}_{\mathsf{prot}}$ as its committed randomness for the protocol executed by $\mathcal{C}_l$.

Upon future calls to the trusted party (the prove phase), every party P_i can prove statements using the secret witness $\boldsymbol{r}_i$. For simplicity we assume that the functionality f is deterministic; randomized functionalities can be made deterministic using standard techniques, by having the trusted party for the setup-commit phase add shares of the same random string (using $(t'+1)$-out-of-n' ECSS) to the members of every committee. The n-party protocol $\pi = \mathsf{Compiler}_{\mathsf{hm}}^{n' \to n}(\pi', \boldsymbol{\mathcal{C}})$ is defined as follows.

Construction 4.15. *(security with abort to fairness with restricted identifiable abort)*

- ***Hybrid Model:*** *The protocol is defined in the Setup-Commit-then-Prove hybrid model with full security.*
- ***Common Input:*** *An n'-party protocol π' and $\boldsymbol{\mathcal{C}} = (\mathcal{C}_1, \ldots, \mathcal{C}_\ell)$. We use the notation P^l_j to refer to the j'th party in $\mathcal{C}_l$.*

- ***Private Input:*** *Every party* P_i*, for* $i \in [n]$*, has private input* $x_i \in \{0,1\}^*$*.*
- ***The Protocol:*** *Let* $\mathcal{J}$ *denote the (initially empty) set of identified corrupted parties. At any point in the protocol, we call a committee* $\mathcal{C}_l$ active *if* $\mathcal{C}_l \cap \mathcal{J} = \emptyset$*.*

1. *Every party* P_i *proceeds as follows:*
   (a) *Call the trusted party to obtain the correlated randomness* $\boldsymbol{r}_i = (r^i_{\mathsf{input}}, \boldsymbol{r}^i_{\mathsf{mask}}, \boldsymbol{r}^i_{\mathsf{prot}})$*.*
   (b) *Broadcast* $x_i \oplus r^i_{\mathsf{input}}$*.*
   (c) *For every active* $\mathcal{C}_l$*, secret share its input as* $(s_i^{1,l}, \ldots, s_i^{n',l}) \leftarrow \mathsf{Share}(x_i)$*, and send* $s_i^{j,l}$ *(masked using the pairwise correlated randomness) to* P^l_j*.*
   (d) *Prove that it sent to every committee shares of its committed input value, using the committed correlated randomness. If a party* P_{i^*} *fails to provide an accepting proof, it is removed from the party set, and if* $i^* \in \bigcup \mathcal{C}_l$ *then* i^* *is added to the list of identified parties* $\mathcal{J}$*.*
2. *For every active* $\mathcal{C}_l$*, every party* $\mathsf{P}^l_j \in \mathcal{C}_l$ *unmasks the vector* $\boldsymbol{s}_{j,l} = (s_1^{j,l}, \ldots, s_n^{j,l})$ *(using* $\boldsymbol{r}_{\mathsf{mask}}$*), and participates in the execution of protocol* π' *for computing* $\mathsf{SS}^{n\to(t',n')}_{\mathsf{in\text{-}out}}(f)$*, over the broadcast channel, where* $\mathsf{P}^l_j \in \mathcal{C}_l$ *uses* $\boldsymbol{s}_{j,l}$ *as its input and its random coins from* $\boldsymbol{r}_{\mathsf{prot}}$*. Denote by* $\boldsymbol{m}_{j,l} = (m_1^{j,l}, \ldots, m_p^{j,l})$ *the messages* $\mathsf{P}^l_j \in \mathcal{C}_l$ *received during the protocol and let* $\mathsf{out}_{j,l}$ *be the output* $\mathsf{P}^l_j \in \mathcal{C}_l$ *received (either a valid value* y *or* $(\perp, i^*)$ *with* $i^* \in \mathcal{C}_l$*).*
3. *For every active* $\mathcal{C}_l$*, every* $\mathsf{P}^l_j \in \mathcal{C}_l$ *broadcasts* $\mathsf{out}_{j,l}$ *and proves to all parties that this is indeed its output value and that* $\mathsf{out}_{j,l}$ *is consistent with the common view and its committed correlated randomness. If party* P_i *receives output values* $(\mathsf{out}_{1,l}, \ldots, \mathsf{out}_{n',l})$ *from a committee* $\mathcal{C}_l$*, where at least* $n'-t'$ *are valid,* P_i *reconstructs the output* y *and outputs it (take the minimal* l *if several committees send valid output values). Otherwise, every committee identified a corrupted party* $\mathsf{P}_{i^*} = \mathsf{P}^l_{j^*} \in \mathcal{C}_l$*; every party adds each such* i^* *to* $\mathcal{J}$*, and outputs* $(\perp, \mathcal{J})$*.*

The simulation follows standard techniques. Given an adversary A, the simulator S first simulates honestly the first call to the Setup-Commit-then-Prove functionality. Next, S commits to zero for every honest party (i.e., sends a random string to A), receives from A commitments to the input values of the corrupted parties, and extracts the values $\{x'_i\}_{i\in\mathcal{I}}$. The simulator S continues by handing A (masked) secret shares of zero on behalf of the honest parties to every corrupted party which is a member on some committee, and verifies that the proof is correct. In addition, S receives from A the values on behalf of the corrupted parties and verifies their validity.

In order to simulate Step 2, the simulator S uses the simulator $\tilde{\mathsf{S}}$ for the ℓ-bounded parallel composition of the protocol π'. For every committee that $\tilde{\mathsf{S}}$ does not abort its computations, S hands secret shares of zero as the output values for the corrupted parties. If there exist committees for which $\tilde{\mathsf{S}}$ did not abort the computation, S sends the values $\{x'_i\}_{i\in\mathcal{I}}$ to the trusted party computing f and receives back the output y. Next, S computes secret sharing of y for every non-aborting committee, and sends them to $\tilde{\mathsf{S}}$ on behalf of honest committee members. In case $\tilde{\mathsf{S}}$ aborts all computation, S updates the set of identified parties $\mathcal{J}$, and hands the trusted party $(\mathsf{abort}, \mathcal{J})$. Finally, S interacts with A according to $\tilde{\mathsf{S}}$ and outputs whatever A outputs. The proof follow in a standard way via the security of the protocol π' and of the ECSS scheme.

To complete the proof of the theorem, we use the fully secure, constant-round implementation of the Setup-Commit-then-Prove functionality in the honest-majority setting, given in [23, Lem. 6.2]. □

4.3 Applications

In Section 4.3.1, we show how to reduce the round complexity for computing Boolean Or from linear to super-constant. In Section 4.3.2, we show how to improve the round complexity in the best-of-both-worlds result of [43]. Finally, in Section 4.3.3, we show how to improve the round complexity in a recent result from [46].

4.3.1 Multiparty Boolean OR

Gordon and Katz [35, Thm. 9] constructed a fully secure protocol that computes the n-party Boolean OR functionality and tolerates an arbitrary number of corruptions. Loosely speaking, every party initially broadcasts a commitment of its input (all parties output 1 if some party does not commit). Next, all parties iteratively run a protocol computing the *committed OR functionality* $f_{\mathsf{com\text{-}or}}$ with identifiable abort, until the result is obtained. The functionality $f_{\mathsf{com\text{-}or}}$ verifies that each party provided the same set of commitments as well as a valid decommitment of its own input. If so the functionality computes Boolean OR; otherwise, it notifies to each party which other parties are not consistent with him, where in the latter case every party continues the protocol only with parties that are consistent with him. In case the protocol outputs $\perp$ and a corrupted party is identified, all parties proceed to the next iteration without the identified party (this can be simulated due to the special properties of the Boolean OR function). It follows that the protocol in [35] requires $t+1$ sequential calls to $f_{\mathsf{com\text{-}or}}$.

The functionality $f_{\mathsf{com\text{-}or}}$

The functionality $f_{\mathsf{com\text{-}or}}$ is run by party-set $\mathcal{P}$ of size n, parametrized by a commitment scheme Com and a vector of commitments $\boldsymbol{c} = (c_1, \ldots, c_n)$.

- Upon the first invocation, with input (x_i, ρ_i) for party $\mathsf{P}_i \in \mathcal{P}$, proceed as follows:
  1. For each party $\mathsf{P}_i \in \mathcal{P}$, if $\mathsf{Com}(x_i; \rho_i) \neq c_i$, add P_i to the (initially empty) set $\mathcal{M}$.
  2. For every $\mathsf{P}_i \in \mathcal{M}$, set $x'_i = 0$; for every $\mathsf{P}_i \in \mathcal{P} \setminus \mathcal{M}$, set $x'_i = x_i$.
  3. Compute $y = x'_1 \vee \ldots \vee x'_n$.
  4. Return $\mathcal{M}$ to all parties and store y as the internal secret state.
- Upon the second invocation, return y to all parties.

Figure 2: The reactive committed OR functionality

We next show how to drastically improve the round complexity of computing Boolean OR, when a constant fraction of parties are honest, by computing $f_{\mathsf{com\text{-}or}}$ with restricted identifiable abort. Note that a direct application of Theorem 4.2 does not help, since the protocol in [35] is

already fully secure and cannot be made secure with fair abort in a meaningful way.[24] The idea is to iteratively invoke $f_{\mathsf{com\text{-}or}}$ with restricted identifiable abort, and eliminate identified corrupted parties. Intuitively, for appropriate parameters, the adversary can abort the computation only in a limited (super-constant) number of invocations, after which the honest parties are guaranteed to obtain the output. However, a closer look shows that *any* party can in fact provide an invalid input to $f_{\mathsf{com\text{-}or}}$ and force the output to be its identity (as it disagrees with all other parties). This will result in a linear number of invocations of $f_{\mathsf{com\text{-}or}}$ before terminating the protocol.

We overcome this obstacle by slightly modifying the definition of $f_{\mathsf{com\text{-}or}}$ from [35]. Initially, we parametrize $f_{\mathsf{com\text{-}or}}$ by the vector of commitments that were sent over the broadcast channel. Second, we would like to ensure that $f_{\mathsf{com\text{-}or}}$ will compute the Boolean OR of the inputs values even if some parties provide invalid decommitments as their inputs. We cannot simply replace the input bit of parties that failed to provide a valid decommitment with 0 (i.e., ignore those parties), since this will lead to the following attack. A commits on behalf of some corrupted P_i to 1 and for all other corrupted parties to 0; upon the first call to $f_{\mathsf{com\text{-}or}}$, it gives an invalid decommitment for P_i and aborts the computation after receiving the output bit by identifying a corrupted P_j. Since the bits of all corrupted parties are treated as 0 in this case, A learns whether all honest parties have 0 or not. Next, $f_{\mathsf{com\text{-}or}}$ is invoked without P_j, and A sends valid decommitments for all corrupted parties; in this case the output will be 1. Clearly, such an attack cannot be simulated in the fully secure ideal model. Therefore, we consider a two-phase single-input reactive version of $f_{\mathsf{com\text{-}or}}$, formally defined in Figure 2. In the first call, the functionality outputs the set of parties that provided invalid decommitments, and stores the result of Boolean OR without those parties as its internal secret state. In the second invocation all parties receive the output bit.

Corollary 4.16 (restating Corollary 1.5)**.** *Assume that TDP, CRH, and non-interactive perfectly binding commitment schemes exist. Then, the n-party Boolean OR functionality can be computed with full security in $\omega(1)$ rounds facing $t = \beta n$ corruptions, for $0 < \beta < 1$.*

Proof. Assume for simplicity of exposition that $n = \Omega(\log(\kappa) \cdot \varphi(\kappa))$. Let $m = \log(\kappa) \cdot \varphi(\kappa)$ with $\varphi(\kappa) = \omega(1)$, $n' = m - \log(\kappa)/\varphi(\kappa)$, and $\ell = \kappa^{\log(e)\cdot\left(\frac{2}{e}+\frac{1}{\varphi(\kappa)}\right)}$. We construct the Boolean OR protocol in the $(f_{\mathsf{elect}}, f_{\mathsf{com\text{-}or}})$-hybrid model, where f_{elect} is computed with full security and $f_{\mathsf{com\text{-}or}}$ with (unfair) $(\ell, n', n'-1)$-identifiable-abort. The corollary will follow from Lemma 2.8 and Theorem 4.6, since under the assumptions in the corollary and using the protocol from [57], the functionality $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f_{\mathsf{com\text{-}or}})$ can be computed in constant rounds with identifiable abort, ℓ times in parallel, facing $n'-1$ corruptions; hence $f_{\mathsf{com\text{-}or}}$ can be computed with $(\ell, n', n'-1)$-identifiable-abort. Note that the reactive functionality $f_{\mathsf{com\text{-}or}}$ receives inputs from the parties only for the first call, and the output of the second call is determined deterministically by the input values to the first call; therefore, following Remark 4.13, the functionality $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f_{\mathsf{com\text{-}or}})$ can indeed be computed with $(\ell, n', n'-1)$-identifiable-abort.

The protocol proceeds as follows with party-set $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$. Initially, every party P_i broadcasts a commitment of its inputs $c_i = \mathsf{Com}(x_i; \rho_i)$; in case some party didn't broadcast, all parties output 1 and halt. Next, denote $\beta' = (1+\beta)/2$, the parties invoke $f_{\mathsf{elect}}(n, m, \beta')$ to elect a committee $\mathcal{C} \subseteq [n]$ of size m. Denote by $\boldsymbol{\mathcal{C}} = (\mathcal{C}_1, \ldots, \mathcal{C}_\ell)$ all subsets of $\mathcal{C}$ of size n' (following Claim 3.11 there are at most ℓ such subsets).

[24] In fact, as pointed out in [22], every fair protocol for Boolean OR can be immediately transformed to a fully secure protocol.

The parties proceed by iteratively invoking the two-phase ideal functionality computing $f_{\mathsf{com\text{-}or}}(\mathcal{P}, \boldsymbol{c})$ with $\boldsymbol{\mathcal{C}}$-identifiable-abort. In the first call, every party P_i sends (x_i, ρ_i) as its input and receives back a subset $\mathcal{M} \subseteq \mathcal{P}$ or $(\mathsf{abort}, \mathcal{J})$; in the latter case P_i sets $\mathcal{P} \setminus \mathcal{J}$ and proceeds to the next iteration. If the first call completed successfully without abort, every party invokes $f_{\mathsf{com\text{-}or}}$ for the second time and receives a bit $y \in \{0,1\}$ or $(\mathsf{abort}, \mathcal{J})$; in the former case party P_i outputs y and halts whereas in the latter case, P_i sets $\mathcal{P} = \mathcal{P} \setminus (\mathcal{J} \cup \mathcal{M})$ and proceeds to the next iteration.

Similarly to [35], the idea behind the simulation is that if the adversary aborts a computation of $f_{\mathsf{com\text{-}or}}$, then it learns new information on the honest parties' inputs only if it sets all the corrupted parties' inputs to 0. Following the definition of $f_{\mathsf{com\text{-}or}}$, the adversary can learn the result using inputs 0 for all corrupted parties only if it committed to zeros in the first round or if it sends invalid decommitments in the first invocation of $f_{\mathsf{com\text{-}or}}$ in some iteration and did not abort it; in both cases A cannot use other inputs and force honest parties to output 1 in later invocations.

Let A be an adversary attacking the protocol in the $(f_{\mathsf{elect}}, f_{\mathsf{com\text{-}or}})$-hybrid model and let $\mathcal{I}$ be the set of corrupted parties. We construct a simulator S for the ideal model computing Boolean OR with full security, as follows. On inputs $\{x_i\}_{i\in\mathcal{I}}$ and auxiliary input z, the simulator S starts by emulating A on these inputs. Initially, S broadcasts commitments to 0 on behalf of honest parties and receives commitments from A on behalf of corrupted parties. In case A didn't send commitments for all corrupted parties, S send to the trusted party 1 on behalf of every corrupted party, outputs whatever A outputs and halts. In the second round, S emulates toward A the committee-election functionality $f_{\mathsf{elect}}(n, m, \beta')$, i.e., S partitions the honest parties to n/m subsets, under the condition that every subset has at least $(1-\beta')n$ parties, hands them to A and receives back a committee $\mathcal{C}$ of size m that contains exactly one of the subsets (and does not intersects the other).

Next, S simulates $f_{\mathsf{com\text{-}or}}$ to A in every iteration as follows:

- **Simulating the first call:**
  - If A sends $(\mathsf{abort}, \mathcal{J})$, the simulator S responds with $(\perp, \mathcal{J})$ and proceeds to the simulation of the next iteration without the parties in $\mathcal{J}$.
  - Otherwise, A provided (x_i, ρ_i) for every corrupted party P_i that was not yet identified. S prepares the list $\mathcal{M}$ of corrupted parties with invalid decommitments, and sends $\mathcal{M}$ to A.
  - If A responds with $(\mathsf{abort}, \mathcal{J})$, the simulator S answers with $(\perp, \mathcal{J})$ and proceeds to the simulation of the next iteration without the parties in $\mathcal{J}$.
  - If A responds with $\mathsf{continue}$, the simulator S proceeds to the simulation of the second call.
- **Simulating the second call:**
  - If A sent $(\mathsf{abort}, \mathcal{J})$, the simulator S responds with $(\perp, \mathcal{J})$ and proceeds to the simulation of the next iteration without the parties in $\mathcal{J} \cup \mathcal{M}$.
  - Otherwise, if one of the inputs (with valid decommitment) sent by A is 1, S responds to A with 1; if all (valid) inputs received from A are 0, S responds to A with the output bit $y \in \{0,1\}$, where in case y is not set yet, S sends to the trusted party 0 on behalf of every corrupted party and receives back the output y.

– If A responds with $(\mathsf{abort}, \mathcal{J})$, the simulator S answers with $(\perp, \mathcal{J})$ and proceeds to the simulation of the next iteration without the parties in $\mathcal{J} \cup \mathcal{M}$.

– If A responds with $\mathsf{continue}$, the simulator S stops simulating $f_{\mathsf{com\text{-}or}}$ toward A.

Next, if S did not send inputs to the trusted party yet, it send 1 on behalf of every corrupted party (and receives back the output 1). Finally, S outputs whatever A outputs and halts.

Proving computational indistinguishability between the output of the honest parties and of A in the execution of the protocol and the output of the honest parties and of S in the fully secure ideal model follows in similar lines to the proof of Gordon and Katz [35]. □

4.3.2 Combining Full Security and Privacy

Given an n-party functionality, the ideal model for computing f with t-full-privacy is defined in a similar way to the ideal model computing f with abort (Definition 2.5) with the exception that the adversary can invoke the trusted party $t+1$ times, and learn the result of f computed on the same inputs for the honest parties and different inputs for the corrupted parties in each time. This notion was introduced by Ishai et al. [43], who showed that assuming the existence of TDP and CRH, for every n-party functionality f there exists a single protocol π that securely computes f with t-full-privacy, tolerating $t < n$ corrupted parties, and achieves t-full-security if $t < n/2$; however, the round complexity in [43] is $O(t)$. We next show how to reduce the round complexity and obtain better privacy guarantees, when a constant number of the parties are honest.

Corollary 4.17 (restating Corollary 1.6)**.** *Assume that TDP, CRH, and non-interactive perfectly binding commitment schemes exist. Let f be an n-party functionality, let $t = \beta n$ for $0 < \beta < 1$, and let $n' = \omega(\log(\kappa))$. Then, there exists a single protocol π, with round complexity $O(n')$, such that:*

1. *π computes f with n'-full-privacy.*
2. *If $t < n/2$ then π computes f with t-full-security.*

Proof sketch. Consider a variant of the functionality $\mathsf{SS}^{n\to n'}_{\mathsf{in}}(f)$ that instead of outputting the output value y in the clear, secret shares y using a $\lceil n'/2 \rceil$-out-of-n' ECSS scheme, denoted $(\mathsf{Share}', \mathsf{Recon}')$. Let π' be a constant-round protocol computing this variant of $\mathsf{SS}^{n\to n'}_{\mathsf{in}}(f)$ with abort (e.g., the protocol from [57]) and let $\mathcal{C} \subseteq [n]$ of size n'. Consider a variant of the compiler presented in Construction 4.8, where in case the computation completes without abort, each party locally reconstructs the shares it received from the committee members in $\mathcal{C}$ and obtains the output value. By adjusting the proof of Theorem 4.6, the protocol $\pi = \mathsf{Compiler}(\pi', \mathcal{C})$ is a constant-round protocol computing f with $\mathcal{C}$-identifiable-abort and $\lfloor (n'-1)/2 \rfloor$-fairness or $(n'-1)$-abort, i.e., if $t' < n'/2$ then π is fair and if $n'/2 \leq t' \leq n'-1$ it is secure with abort, and in case of abort a corrupted party in $\mathcal{C}$ is identified. □

4.3.3 Partially Identifiable Abort to Full Security

Ishai et al. [46] introduced security with α-partially identifiable abort as security with identifiable abort, such that upon abort, a subset of parties is identified, where at least an α-fraction of the subset is corrupted. Next, they presented a transformation, in the honest-majority setting, from α-partially identifiable abort, for $\alpha \leq 1/2$, to full security. The transformation is based on the

player-elimination approach, and the idea is to compute $\mathsf{SS}_{\mathsf{in\text{-}out}}^{n\to(t',n')}(f^n)$ with α-partially identifiable abort by a committee that initially consists of all the parties (i.e., $n'=n$), where in case of abort, all the identified parties (both honest and corrupted) are removed from the committee. It follows that the number of iterations is $O(n)$. We next show how to reduce the round complexity when a constant fraction of the parties are honest.

Corollary 4.18 (restating Corollary 1.7)**.** *Let f^n be an n-party functionality, let $n'=\log^*(\kappa)\cdot\log(\kappa)$, let $0<\beta<\beta'<1/2$, let $t=\beta n$, let $t'=\beta' n'$, and let π' be an r'-round protocol that securely computes $\mathsf{SS}_{\mathsf{in\text{-}out}}^{n\to(t',n')}(f^n)$ with β'-partially identifiable abort, tolerating t' corruptions. Then, f^n can be computed with full security, tolerating t corruptions, by a $O(t'\cdot r')$-round protocol that uses the protocol π' in a black-box way.*

Proof sketch. The proof follows in similar lines to proof of Theorems 4.2 and 4.14, with the exception that upon abort in a committee, all identified parties are removed from the committee. In more detail, the parties initially elect a committee $\mathcal{C}$ of size $\log(\kappa)\cdot\log^*(\kappa)$. Next, every party secret shares its input to the committee members, that execute the protocol π'. Upon completion, every committee member broadcasts the output value it received. If the output consists of at most t' messages of the form $(\perp,\cdot)$, every party reconstructs the output value and halts. Otherwise, if more than t' messages are of the form $(\perp,\mathcal{J})$, the parties consider the subset $\mathcal{J}$ that appears in most messages, set $\mathcal{C}=\mathcal{C}\setminus\mathcal{J}$, and re-iterate. □

4.4 Computation Over Shared Inputs

In this section, we provide the definitions of the functionalities $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f)$ and $\mathsf{SS}_{\mathsf{in\text{-}out}}^{n\to(t',n')}(f)$ that were used earlier.

A basic idea underlying our reductions is to delegate the computation to a small committee. Intuitively, this is done by having each party secret share its input value among the committee members, and letting the committee compute the function over the shared inputs. The type of the secret-sharing scheme that we use depends on whether an honest majority is assumed or not. In the honest-majority setting, we consider an error-correcting secret-sharing scheme (ECSS, see Definition 2.2), allowing honest committee members to reconstruct the correct input values even if corrupted committee members arbitrarily modify the shares they received.

The case that an honest majority cannot be guaranteed is somewhat more subtle. In this case, ECSS schemes do not exist [44], and we need to use an n'-out-of-n' secret sharing. Furthermore, to prevent corrupted parties from computing the function on wrong inputs (by changing some of their shares), we let each party broadcast commitments to its shares. The functionality computed by the committee, parametrized by all the commitments, first verifies that the decommitments are valid and only later reconstructs the input values and evaluates the function.

The Verify-Reconstruct-Compute functionality. Given a public-output n-party functionality f and $n'<n$, the n'-party functionality $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f)$ is parametrized by n vectors of commitments $\boldsymbol{c}_i=(c_i^1,\ldots,c_i^{n'})$, for $i\in[n]$. Each of the commitments c_i^j commits a unique value s_i^j; denote $x_i=\bigoplus_{j\in[n']}s_i^j$. Each input value to $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f)$ consists of a vector of n values, such that the i'th value for the j'th party is the decommitment to c_i^j. The functionality validates that all of the decommitments are valid, reconstructs all x_i's, evaluates $y=f(x_1,\ldots,x_n)$ and outputs the result y to all n' parties. A formal description of the functionality appears in Figure 3.

The functionality $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f)$

The n'-party functionality $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f)$ is parametrized by an n-party public-output functionality f, a non-interactive commitment scheme Com and vectors of commitments $\boldsymbol{c}_i = (c_i^1, \ldots, c_i^{n'})$, for every $i \in [n]$. $\mathsf{SS}_{\mathsf{in}}^{n\to n'}(f)$ is formally defined as follows, on inputs $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{n'})$.

1. For every $j \in [n']$, parse $\boldsymbol{s}_j$ as an n-vector $(\hat{s}_1^j, \ldots, \hat{s}_n^j)$; for every $i \in [n]$, parse $\hat{s}_i^j$ as a pair $\hat{s}_i^j = (s_i^j, \rho_i^j)$ and verify that $c_i^j = \mathsf{Com}(s_i^j; \rho_i^j)$. If there exists a malformed $\boldsymbol{s}_j$, output $(\perp, j)$ to all parties (for the smallest such j).
2. For every $i \in [n]$, compute $x_i = \bigoplus_{j\in[n']} s_i^j$.
3. Compute $y = f(x_1, \ldots, x_n)$ and output y to all parties.

Figure 3: The Verify-Reconstruct-Compute functionality

The Reconstruct-Compute-Share functionality This functionality is parametrized by a $(t'+1)$-out-of-n' error-correcting secret-sharing scheme (ECSS, Definition 2.2) that can correct up to t' errors during the reconstruction procedure. Given an n-party functionality f, denote by $\mathsf{SS}_{\mathsf{in\text{-}out}}^{n\to(t',n')}(f)$ the n'-party functionality, that receives as input secret shares of an n-tuple $(x_1, \ldots, x_n)$, evaluates f on the reconstructed values as $y = f(x_1, \ldots, x_n)$,[25] and outputs secret shares of y. A formal description of the functionality appears in Figure 4.

The functionality $\mathsf{SS}_{\mathsf{in\text{-}out}}^{n\to(t',n')}(f)$

The n'-party functionality $\mathsf{SS}_{\mathsf{in\text{-}out}}^{n\to(t',n')}(f)$, parametrized by an n-party functionality f and a $(t'+1)$-out-of-n' ECSS scheme $(\mathsf{Share}, \mathsf{Recon})$, is formally defined as follows, on inputs $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{n'})$.

1. For every $j \in [n']$ parse $\boldsymbol{s}_j$ as an n-vector $(s_1^j, \ldots, s_n^j)$ (in case $\boldsymbol{s}_j$ cannot be parsed, set to the vector $(0, \ldots, 0)$; the value 0 is arbitrary).
2. For every $i \in [n]$, compute $x_i = \mathsf{Recon}(s_i^1, \ldots, s_i^{n'})$ (in case the reconstruction fails, set $x_i = \tilde{x}_i$ for some default value $\tilde{x}_i$).
3. Compute $y = f(x_1, \ldots, x_n)$.
4. Secret share the result $(y_1, \ldots, y_{n'}) \leftarrow \mathsf{Share}(y)$.
5. Set the output for every $j \in [n']$ to be y_j.

Figure 4: The Reconstruct-Compute-Share functionality

5 Necessity of Super-Constant Sequential Fair Calls

In this section, we prove that a super-constant number of *functionality rounds*, in which (possibly parallel) calls to the fair functionality are made, is necessary for a fully secure implementation of

[25] For the sake of simplicity, we consider f as a public-output functionality. Adjusting the protocol for private-output functionalities is straightforward.

some functionalities. Specifically, we show the necessity of such number of functionality rounds when the functionality in consideration is coin flipping.

The model in which the lower bound is proved is defined in Section 5.1, and the lower bound is stated and proved in Section 5.2. The lower bound proof uses a useful corollary of Cleve's lower bound [21], whose proof is given in Section 5.3.

5.1 The Model

Consider an n-party coin-flipping protocol π executed in the hybrid model where parties can compute f_{cf}^n (n-party coin-flipping functionality, see Definition 3.20) with fairness and $\mathcal{C}$-id-abort for every $\mathcal{C} \subseteq [n]$ (see Definition A.1). Namely, a protocol in this model has in addition to the standard communication rounds, also *functionality rounds* in which any subset (committee) $\mathcal{C} \subseteq [n]$ of the parties can ask the trusted party to flip a coin, and the trusted party outputs a uniform bit visible to all n parties. If the subset $\mathcal{C}$ contains a corrupted party, then the attacker can abort this call before learning the value of the output bit, but at the price of revealing the identity of a corrupted party in the committee to all parties (even those not in the committee). An all-corrupted committee can determine the output of the trusted party arbitrarily, without being identified.

We prove the lower bound in a stronger hybrid model than the above (where the life of the honest parties are easier) that allows *parallel* calls to the trusted party by different committees at the same functionality round. In such a case, (only) an all-corrupted committee is assumed to be rushing, and can decide whether to abort or not based on the outcome of other parallel calls. The output of all other committees, is published at the same time to all parties. We denote the above hybrid model by $(f_{\mathsf{cf}}^n, [n]\text{-id-abort-par})$-*hybrid model*. An s-*call* protocol in this model has at most s functionality rounds.

Recall that in a coin-flipping protocol, the output of an all-honest execution is a common uniform bit. The protocol is γ-*consistent* if in an honest execution, any two parties output the same value with probability at least $1/2 + \gamma$ (in the case that $\gamma = 1/2 - \mathrm{neg}(\kappa)$, we simply say the protocol is consistent). Finally, a fail-stop attacker for π might only deviate from the protocol by early aborting (in the $(f_{\mathsf{cf}}^n, [n]\text{-id-abort-par})$-hybrid model, this can be done either by stop sending messages during an interaction round, or by aborting a call to the ideal functionality).

Remark 5.1 (Is this the right model?)**.** *The above model is generous with the honest parties, as it assumes that* only *an all-corrupted committee can be rushing: only a party in such a committee can decide to abort or not based on the result of the other parallel calls to the trusted party. Indeed, in this model, the reductions of Section 3.1 can be modified, in the spirit of the reductions given in Section 3.2, to yield protocols of only $\omega(1)$ sequential calls to the fair functionality.*

We do not know whether the above model can be justified by the existence of a fully secure protocol that computes the functionality at hand (at least not for a randomized functionality in the dishonest-majority setting). Hence, the model we actually use to prove our positive results in Section 3.1 is much more pessimistic, and essentially does not allow parallel invocations of the trusted party. In this pessimistic model, it is not hard to prove that a super-logarithmic number of functionality rounds is needed, making the result of Section 3.1 optimal in this respect.

5.2 The Lower Bound

Given the above formulation, our lower bound is stated as follows.

Theorem 5.2. *Let π be an n-party, s-call, m-time, γ-consistent coin-flipping protocol in the $(f_{\mathsf{cf}}^n, [n]\text{-}\mathsf{id\text{-}abort\text{-}par})$-hybrid model. Then, there exists a fail-stop attacker, controlling a βn-size subset of the parties, for $1/2 < \beta < 1$, that can bias the output of π by $\Omega(\gamma n^{-c}/m)$, for $c = O(s \cdot \log m / \log n)$.*

The above yields that for a constant s and non-negligible γ, protocol π is not fully secure facing up to βn corruptions. For simplicity, we will prove the theorem for consistent protocols (i.e., $\gamma = 1/2$), but our technique readily captures any non-negligible γ.

The proof of Theorem 5.2 follows the high-level description given in the Introduction (Section 1.2). In Section 5.2.1, we show how to bias any coin-flipping protocol, assuming the ideal functionality is *not* invoked by "large" committees (all committees are of size at most $c \cdot \log n$, for a constant c to be determined by the analysis). In Section 5.2.2, we adjust the proof of Section 5.2.1 for proving Theorem 5.2, by showing that large committees are not useful for protocols of constant number of functionality rounds.

5.2.1 Biasing Protocols with No Large Committees

In this section, we prove Theorem 5.2 assuming the ideal functionality is only invoked by "small" committees. This is captured by the following lemma.

Lemma 5.3. *Let π be an n-party, s-call, m-time, consistent coin-flipping protocol in the $(f_{\mathsf{cf}}^n, [n]\text{-}\mathsf{id\text{-}abort\text{-}par})$-hybrid model, in which all calls to the ideal functionality are made by committees of size at most $c \cdot \log n$ for a constant c. Then, there exists a fail-stop attacker, controlling a βn-size subset of the parties, for $1/2 < \beta < 1$, that can bias the output of π by $\Omega(n^{-c}/m)$.*

The proof is a reduction of the multiparty coin-flipping protocol π in the $(f_{\mathsf{cf}}^n, [n]\text{-}\mathsf{id\text{-}abort\text{-}par})$-hybrid model to a two-party coin-flipping protocol in the plain model. Since the latter can be biased by a fail-stop attacker [21], the proof is completed by showing how to translate any such fail-stop attacker into a (fail-stop) attacker on π. Interestingly, the proof of this part holds for any (no-large-committees) protocol, regardless of the number of its functionality rounds. The following discussion is with respect to a fixed $(1-\beta)n$-size subset $\mathcal{S} \subseteq [n]$ (chosen arbitrarily).

Loosely speaking, the two-party protocol $\psi = (\mathsf{P}_0, \mathsf{P}_1)$ relates to the n-party protocol π as follows: the n parties of π are partitioned into the two subsets $\mathcal{S}$ and $[n] \setminus \mathcal{S}$, that are controlled by P_0 and P_1, respectively, in a random emulation of π. Party P_0 emulates the ideal functionality: when ℓ committees, $\mathcal{C}_1, \ldots, \mathcal{C}_\ell$, call the ideal functionality in parallel in π, this translates in ψ to P_0 sending ℓ uniformly distributed bits, sequentially, one in each round. These bits are interpreted by the parties as the answers of the ideal functionality. If, as a concrete example, P_0 aborts in a round in which it is supposed to send a uniform bit, emulating the answer of the ideal functionality to a call by some committee $\mathcal{C}$, party P_1 interprets it as if the first party in $\mathcal{C}$ aborted the call and continues the execution of π as follows: it randomly chooses some $(1-\beta)n$-size subset of the parties under its control that does not intersect $\mathcal{C}$, denoted $\mathcal{T}$, and emulates the remainder of the execution as if all parties outside $\mathcal{T}$ abort right after the call to the trusted party. For technical reasons, it would have been convenient to assume that $\mathcal{C}$ has *exactly* $c \log n$ elements outside $\mathcal{S}$. Since the latter might not always be the case, when choosing the subset $\mathcal{T}$, party P_1 first appends arbitrary elements to $\mathcal{C}$ so that the resulting subset has the requited property.

We now formally define the two-party protocol ψ induced by the multiparty protocol π. For $\mathcal{C} \subseteq [n]$, let $\mathsf{extend}(\mathcal{C})$ be the committee $\mathcal{C}'$ obtained from $\mathcal{C}$ by padding it with arbitrary (but fixed) parties from $[n] \setminus \mathcal{S}$, such that $|\mathcal{C}' \setminus \mathcal{S}| = c \log n$ (if $|\mathcal{C} \setminus \mathcal{S}| = c \log n$, then $\mathcal{C}' = \mathcal{C}$). The two-party coin-flipping protocol in the standard model $\psi = (\mathsf{P}_0, \mathsf{P}_1)$ is defined as follows.

Protocol 5.4 (Protocol $\psi = (\mathsf{P}_0, \mathsf{P}_1)$: two-party coin flipping from multiparty coin flipping)**.**

1. *Party P_0 controls the parties in $\mathcal{I}_0 = \mathcal{S}$ and P_1 controls the parties in $\mathcal{I}_1 = [n] \setminus \mathcal{S}$ in an emulation of π: in each communication round of π, party P_i (for $i \in \{0,1\}$) internally emulates all messages among parties in $\mathcal{I}_i$. Party P_i also sends to P_{1-i} all messages from parties in $\mathcal{I}_i$ to parties in $\mathcal{I}_{1-i}$ and over the broadcast channel.*

2. *Party P_0 emulates the ideal functionality: when a parallel call to the functionality by ℓ committees, denoted by $\mathcal{C}_1, \ldots, \mathcal{C}_\ell$,[26] is due, P_0 sends ℓ separate messages, one for each committee, each containing a uniformly distributed bit.*

Output: *Party P_0 (resp., P_1) outputs the output value of an arbitrary non-aborted party in $\mathcal{I}_0$ (resp., $\mathcal{I}_1$) in π.*

Dealing with abort:

1. Party P_i aborts while emulating a communication round: *party P_{1-i} continues the emulation internally, as if all the parties in $\mathcal{I}_i$ stopped sending messages in the current round of π. If the aborting party is P_0, then P_1 continues the execution by emulating all remaining calls to the ideal functionality on its own.*

2. Party P_0 aborts while emulating a functionality round: *let $\mathcal{C}_1, \ldots, \mathcal{C}_\ell$ be the committees of the current functionality round, and let $\mathcal{C}_j$ be the committee corresponding to the aborting round in ψ. Party P_1 continues the emulation as if the party with the minimal index in $\mathcal{C}_j$ aborted the call to the ideal functionality (in π), and flips the remaining coins for $\mathcal{C}_{j+1}, \ldots, \mathcal{C}_\ell$ internally. Next, P_1 decides on its output as follows: P_1 chooses uniformly at random a $(1-\beta)n$-size subset $\mathcal{T} \subseteq [n] \setminus (\mathcal{S} \cup \mathsf{extend}(\mathcal{C}_j))$, and continues the emulation of π internally as if all remaining parties outside $\mathcal{T}$ stopped sending messages.*

In the analysis of the attack on π given below, it will be helpful to think of the subset of corrupted parties as a randomly chosen βn-size subset containing $\mathcal{S}$, and then bound the probability that it strictly contains $\mathsf{extend}(\mathcal{C})$ (where $\mathcal{C}$ is the aborted committee). The set $\mathcal{T}$ will than play the role of the honest parties, allowing us to easily relate the expected bias (over the choice of $\mathcal{T}$) of our attacker for π to that of an attacker controlling P_0 in ψ.

We prove the lemma by translating the fail-stop attacker for ψ guaranteed by [21] (see Lemma 5.5) into a fail-stop attacker for π. The fail-stop attacker for the two-party protocol naturally translates into an attack on π, if the aborting round in not a functionality round (simply aborts the parties controlled by the corrupted party). For the case that the aborting round is a functionality round, we exploit the fact that, by assumption, all committees are small. Assume the guaranteed attacker for ψ corrupts party P_0 and attacks by aborting in a round in which P_0 is due to emulate the answer of the ideal functionality to some committee $\mathcal{C}^*$. In order to mimic such an

[26] The order of the committees is set arbitrarily.

attack in π, the attacker needs to control all parties in $\mathcal{C}^*$. This is indeed the case with noticeable probability when it chooses the set $\mathcal{T}$ at random.

Analyzing the above is rather subtle, and crucially relies on the fact that $\mathcal{T}$ is randomly chosen by P_1 *during* the execution of ψ. This way, the aborting round in ψ (which is pre-determined and independent of a particular execution of the protocol) is fixed regardless of the choice of $\mathcal{T}$. Hence, the identity of the parties in $\mathcal{C}^*$ is independent of the choice of $\mathcal{T}$ in ψ, and therefore, independent of the choice of the corrupted parties (outside $\mathcal{S}$) in π. In case we would have fixed $\mathcal{T}$ in advance, and let P_0 control all parties in $[n] \setminus \mathcal{T}$, it might have been the case that for any choice of $\mathcal{T}$, the aborting round in ψ would have corresponded to a committee $\mathcal{C}^* \subseteq \mathcal{T}$. In this case, an attacker that controls P_0 and aborts during a functionality round cannot be translated to an attacker on π.

When formalizing the above, we use a variant of the two-party attacker guaranteed by Cleve [21]. Specifically, we assume the fail-stop attacker either aborts in a predetermined round, or does not abort at all. On the other hand, Cleve's attacker either aborts in a given predetermined round, or aborts in the next round. The existence of the required attacker is stated in the following lemma, whose proof (see Section 5.3) easily follows the result of Cleve [21].

Lemma 5.5. *Let $\psi = (\mathsf{P}_0, \mathsf{P}_1)$ be a two-party, r-round, γ-consistent coin-flipping protocol. Then, there exist a party $\mathsf{P} \in \{\mathsf{P}_0, \mathsf{P}_1\}$, a round index $i^* \in [r]$, an index $j^* \in \{i^*, i^* - 1\}$, and a bit $b \in \{0, 1\}$ such that the following holds. Let b_{j^*} denote the output of P in case the other party aborts in round j^*, and consider the fail-stop attacker A, that corrupts P and instructs P to abort in round i^* if $b_{j^*} = b$. Then, A biases the output of the other party by $\gamma/(8r + 2)$.*

Let A^ψ be the fail-stop attacker for ψ guaranteed by Lemma 5.5. Since ψ consists of at most m communication rounds (recall that m is a bound on the running time of π), by Lemma 5.5, A^ψ biases the output of ψ by at least $1/(16m + 4)$. Observe that if A^ψ corrupts P_1, an attacker for π controlling $\mathcal{I}_1 = [n] \setminus \mathcal{S}$ can bias the output of the honest parties by the same bias, by simply mimicking A^ψ (acting honestly until some predetermined round i, and then aborting with all of the parties in $\mathcal{I}_1$ only if A^ψ aborted in the corresponding round). Similarly, if A^ψ corrupts P_0 and attacks by aborting in a round that corresponds to an interaction round of π, then a similar attack is also possible. Hence, we assume without loss of generality that A^ψ attacks by corrupting P_0, and aborting, if at all, in a round of ψ that corresponds to a functionality round in π.

Let i^* be the round in ψ in which A^ψ might abort, let $\mathcal{C}^*$ be the committee in π that corresponds to round i^* in ψ, and let i^*_f be the index of the functionality round in π in which $\mathcal{C}^*$ calls the ideal functionality. For a $(1 - \beta)n$-size subset $\mathcal{T} \subseteq [n] \setminus \mathcal{S}$, the attacker $\mathsf{A}^\pi_\mathcal{T}$, corrupting the parties in $[n] \setminus \mathcal{T}$, is defined as follows.

Algorithm 5.6 (The fail-stop attacker $\mathsf{A}^\pi_\mathcal{T}$ for π)**.**

1. *Start emulating A^ψ, by forwarding to A^ψ the messages sent by the honest party in ψ and the results of the calls to the ideal functionality.*[27]
2. *Until the i^*_f'th functionality round, act honestly.*
3. *In the i^*_f'th functionality round:*

[27] This is done to ensure the coins that P_0 flips in ψ are the same as the coins flipped by the ideal functionality in the execution of π. Note that this does not change the distribution over the flipped bits.

(a) *If* $\mathsf{extend}(\mathcal{C}^*) \subseteq [n] \setminus \mathcal{T}$*: wait until all other committees but* $\mathcal{C}^*$ *in the* i_f^**'th functionality round are done. Let* $\mathcal{C}_1, \ldots, \mathcal{C}_\ell$ *be the committees that call the ideal functionality in the* i_f^**'th functionality round of* π*, and correspond to rounds earlier than* i^* *in* ψ*. Give the result bits of* $\mathcal{C}_1, \ldots, \mathcal{C}_\ell$ *to* A^ψ*, sequentially. If* A^ψ *chooses to abort, instruct the minimal-index party in* $\mathcal{C}^*$ *to abort the call to the ideal functionality,*[28] *and in the next interaction round of* π*, instruct all parties in* $[n] \setminus \mathcal{T}$ *to abort. If* A^ψ *does not abort, complete the honest execution of* π*.*

(b) *If* $\mathsf{extend}(\mathcal{C}^*) \not\subseteq [n] \setminus \mathcal{T}$*: continue in the honest execution of* π*.*

Namely, the fail-stop attacker $\mathsf{A}^\pi_\mathcal{T}$, corrupting the parties in $[n] \setminus \mathcal{T}$, uses the real interaction of π to emulate an interaction of $(\mathsf{A}^\psi, \mathsf{P}_1)$. The attacker $\mathsf{A}^\pi_\mathcal{T}$ acts honestly until (the emulated) A^ψ chooses to abort, which, by assumption, happens in a functionality round. Recall that in the emulation of π done in ψ, such an abort is interpreted as the first member of the corresponding committee aborting the call to the ideal functionality, followed by all parties outside a random $(1-\beta)n$-size subset of $[n] \setminus (\mathcal{S} \cup \mathsf{extend}(\mathcal{C}^*))$, aborting after the call. The above attack goes through, only if all parties in $\mathsf{extend}(\mathcal{C}^*)$ are corrupted. The key observation is that for a randomly chosen $\mathcal{T}$, with noticeable probability, the set $[n] \setminus \mathcal{T}$ contains this (small, by assumption) committee. If the latter happens, then $\mathsf{A}^\pi_\mathcal{T}$ perfectly replicates the attack by A^ψ, and thus incurs the same bias in π as A^ψ achieves in ψ. Hence, there exists some $\mathcal{T}$ such that $\mathsf{A}^\pi_\mathcal{T}$ succeeds in biasing the protocol.

The above intuition is made rigorous in the following lemma that concludes the proof of Theorem 5.2 for the no-large-committees case.

Lemma 5.7. *There exists a* $(1-\beta)n$*-size subset* $\mathcal{T} \subseteq [n] \setminus \mathcal{S}$ *such that* $\mathsf{A}^\pi_\mathcal{T}$ *biases the output of the honest parties in* π *by* $\Omega(n^{-c}/m)$*.*

Proof. Recall our notation: let i^* be the round in ψ in which A^ψ might abort, let $\mathcal{C}^*$ be the committee in π that corresponds to round i^* in ψ, and let i_f^* be the index of the functionality round in π in which $\mathcal{C}^*$ calls the ideal functionality. Denote $\widehat{\mathcal{C}^*} = \mathsf{extend}(\mathcal{C}^*)$ and hence, by construction, $\left|\widehat{\mathcal{C}^*} \setminus \mathcal{S}\right| = c \log n$. Finally, recall that $\mathcal{T} \subseteq [n] \setminus \mathcal{S}$ and that $\mathsf{A}^\pi_\mathcal{T}$ corrupts the parties in $[n] \setminus \mathcal{T}$.

Let $B_\mathcal{T}$ denote the bias of the output of the honest parties in π, in a random execution of π in which $\mathsf{A}^\pi_\mathcal{T}$ controls the parties in $[n] \setminus \mathcal{T}$. Let V denote the view of the parties in $\mathcal{S}$ up to the i_f^* functionality round, including the answers for the calls made in this round by the committees appearing before $\mathcal{C}^*$ in the (arbitrary) order of ψ. It holds that the expected bias of the honest parties can be written as follows:

$$\begin{aligned}
\mathrm{E}_\mathcal{T}\left[\mathrm{E}_v\left[\mathrm{E}\left[B_\mathcal{T} \mid V=v\right]\right]\right] &= \mathrm{E}_v\left[\mathrm{E}_\mathcal{T}\left[\mathrm{E}\left[B_\mathcal{T} \mid V=v\right]\right]\right] \\
&= \mathrm{E}_v\left[\mathrm{E}_{\mathcal{T} \mid \mathcal{T} \cap \widehat{\mathcal{C}^*} = \emptyset}\left[\mathrm{E}\left[B_\mathcal{T} \mid V=v\right]\right] \cdot \Pr\left[\mathcal{T} \cap \widehat{\mathcal{C}^*} = \emptyset \mid V=v\right]\right] \\
&+ \mathrm{E}_v\left[\mathrm{E}_{\mathcal{T} \mid \mathcal{T} \cap \widehat{\mathcal{C}^*} \neq \emptyset}\left[\mathrm{E}\left[B_\mathcal{T} \mid V=v\right]\right] \cdot \Pr\left[\mathcal{T} \cap \widehat{\mathcal{C}^*} \neq \emptyset \mid V=v\right]\right] \\
&= \mathrm{E}_v\left[\mathrm{E}_{\mathcal{T} \mid \mathcal{T} \cap \widehat{\mathcal{C}^*} = \emptyset}\left[\mathrm{E}\left[B_\mathcal{T} \mid V=v\right]\right]\right] \cdot \Pr\left[\mathcal{T} \cap \widehat{\mathcal{C}^*} = \emptyset\right] \\
&+ \mathrm{E}_v\left[\mathrm{E}_{\mathcal{T} \mid \mathcal{T} \cap \widehat{\mathcal{C}^*} \neq \emptyset}\left[\mathrm{E}\left[B_\mathcal{T} \mid V=v\right]\right]\right] \cdot \Pr\left[\mathcal{T} \cap \widehat{\mathcal{C}^*} \neq \emptyset\right].
\end{aligned} \tag{3}$$

[28] Note that this is indeed possible in the case in hand; since $\mathcal{C}^*$ is all-corrupted, and we assumed $\mathsf{A}^\pi_\mathcal{T}$ to be rushing. Thus, it can give A^ψ all results to calls made by $\mathcal{C}_1, \ldots, \mathcal{C}_\ell$, and only then choose to abort the call by $\mathcal{C}^*$.

The above expectations are with respect to $v \leftarrow V$ and a uniformly distributed set $\mathcal{T} \leftarrow [n] \setminus \mathcal{S}$ of size $(1-\beta)n$. The first equality follows by the fact that V is independent of the choice of $\mathcal{T}$ (since $\mathcal{T}$ is always disjoint of $\mathcal{S}$). The last equality holds since, by construction, the size of $\widehat{\mathcal{C}^*} \setminus \mathcal{S}$ is fixed, and thus the event $\mathcal{T} \cap \widehat{\mathcal{C}^*} = \emptyset$ is independent of V.

By the definition of the two-party protocol ψ and the attacker $\mathsf{A}^{\pi}_{\mathcal{T}}$, and due to the fact that $\widehat{\mathcal{C}^*}$ has a fixed number of elements in $[n] \setminus \mathcal{S}$, it holds that the expected bias of the honest parties in π conditioned on $\widehat{\mathcal{C}^*}$ being all corrupted, $\mathrm{E}_v\left[\mathrm{E}_{\mathcal{T}|\mathcal{T}\cap\widehat{\mathcal{C}^*}=\emptyset}\left[\mathrm{E}\left[B_{\mathcal{T}} \mid V=v\right]\right]\right]$, is equal to the bias of the honest party in ψ in an execution with A^{ψ}. Hence,

$$\mathrm{E}_v\left[\mathrm{E}_{\mathcal{T}|\mathcal{T}\cap\widehat{\mathcal{C}^*}=\emptyset}\left[\mathrm{E}\left[B_{\mathcal{T}} \mid V=v\right]\right]\right] \geq 1/(16m+4). \tag{4}$$

Since when $\mathcal{T}\cap\widehat{\mathcal{C}^*} \neq \emptyset$, the attacker $\mathsf{A}^{\pi}_{\mathcal{T}}$ simply acts as an honest P_0 would, and for similar reasons as in the case of Equation (4), the expectation $\mathrm{E}_v\left[\mathrm{E}_{\mathcal{T}|\mathcal{T}\cap\widehat{\mathcal{C}^*}\neq\emptyset}\left[\mathrm{E}\left[B_{\mathcal{T}} \mid V=v\right]\right]\right]$ is the expected bias of the output of the honest parties in an *honest execution* of π. Hence,

$$\mathrm{E}_v\left[\mathrm{E}_{\mathcal{T}|\mathcal{T}\cap\widehat{\mathcal{C}^*}\neq\emptyset}\left[\mathrm{E}\left[B_{\mathcal{T}} \mid V=v\right]\right]\right] = 0. \tag{5}$$

It follows that

$$\mathrm{E}_{\mathcal{T}}\left[\mathrm{E}_v\left[\mathrm{E}\left[B_{\mathcal{T}} \mid V=v\right]\right]\right] = \frac{1}{16m+4} \cdot \Pr\left[\mathcal{T}\cap\widehat{\mathcal{C}^*}=\emptyset\right]. \tag{6}$$

Finally, a simple counting argument yields that:

$$\Pr\left[\mathcal{T}\cap\widehat{\mathcal{C}^*}=\emptyset\right] = \frac{\binom{(2\beta-1)n}{c\log n}}{\binom{\beta n}{c\log n}} \geq \left(\frac{2\beta-1}{\beta e}\right)^{c\log n}. \tag{7}$$

Putting everything together, we conclude that

$$\mathrm{E}_{\mathcal{T}}\left[\mathrm{E}_v\left[\mathrm{E}\left[B_{\mathcal{T}} \mid V=v\right]\right]\right] \geq \frac{1}{16m+4}\left(\frac{2\beta-1}{\beta e}\right)^{c\log n} \in \Omega(n^{-c}/m). \tag{8}$$

In particular, there exists a set $\mathcal{T} \in [n]\setminus\mathcal{S}$ such that $\mathsf{A}^{\pi}_{\mathcal{T}}$ biases the output of the honest parties in π by $\Omega(n^{-c}/m)$. □

5.2.2 Biasing Arbitrary Protocols

In this section, we extend the approach of Section 5.2.1 to the case where there may be large committees (i.e., larger than $c\log n$). Hence, proving Theorem 5.2.

Loosely speaking, the two-party protocol and the attack of Section 5.2.1 are adjusted in the following manner. The revised two-party protocol ψ also includes s linear-size subsets of parties $\mathcal{J}_1, \ldots, \mathcal{J}_s$, each associated with a single functionality round. In the i'th functionality round of the emulated execution of π, parties P_0 and P_1 go on as if the parties in $\mathcal{J}_i$ abort all calls to the ideal functionality by committees that intersect with $\mathcal{J}_i$. The attacker $\mathsf{A}^{\pi}_{\mathcal{T}}$ we construct for π, corrupting all parties in $[n] \setminus \mathcal{T}$, acts as the attacker from the no-large-committees case (Algorithm 5.6), with the following additional change: in the i'th functionality round, it instructs the parties in $\mathcal{J}_i$ to prematurely abort all calls made by committees it intersects with (we make sure $\mathcal{J}_1, \ldots, \mathcal{J}_s \subseteq$

$[n] \setminus \mathcal{T}$). Since the subsets $\mathcal{J}_1, \ldots, \mathcal{J}_s$ are all of linear size, with high probability the above strategy will abort all calls made by committees of size larger than $c \log n$, essentially leaving us with a protocol with no large committees, and thus vulnerable to the strategy of Algorithm 5.6.

Moving to the formal proof, let $\beta' = (\beta - 1/2)/s$, and let $c = \log(m(32m+10))/\beta' \log n$. The following claim shows that there exists a collection $\mathcal{J}_1, \ldots, \mathcal{J}_s$ such that in a random execution of π, for every $i \in [s]$, subset $\mathcal{J}_i$ intersects all committees of size at least $c \log n$ of the i'th functionality round with high probability.

Claim 5.8. *For s distinct subsets $\overline{\mathcal{J}} = \mathcal{J}_1, \ldots, \mathcal{J}_s$, let $\pi_{\overline{\mathcal{J}}}$ denote the variant of π in which in the i'th functionality round, the parties in $\mathcal{J}_i$ abort all functionality calls made by committees they take part in this functionality round. Let $E^i_{\overline{\mathcal{J}}}$ denote the event, defined with respect to a random execution of $\pi_{\overline{\mathcal{J}}}$, that a call to the trusted party is made by a committee $\mathcal{C}$ of size larger than $c \log n$ in the i'th functionality round, and $\mathcal{C} \cap \mathcal{J}_i = \emptyset$. Then, there exists a collection $\overline{\mathcal{J}} = \mathcal{J}_1, \ldots, \mathcal{J}_s \subseteq [n]$ of distinct $\beta' n$-size subsets, such that $\Pr\left[E^i_{\overline{\mathcal{J}}}\right] \leq 1/(32m+10)$ for every $i \in [s]$.*

Proof. The claim is proved using a simple probabilistic argument. Let $\overline{\mathcal{J}} = \mathcal{J}_1, \ldots, \mathcal{J}_s$ be a collection of s disjoint β'-subsets of $[n]$ chosen uniformly at random. We show that $\Pr\left[E^i_{\overline{\mathcal{J}}}\right] \leq 1/(32m+10)$ for all $i \in [s]$, where the probability is taken over the selection of $\mathcal{J}_1, \ldots, \mathcal{J}_s$ and the randomness of the parties and the ideal functionality. The claim thus immediately follows.

Let $i \in [s]$ and let $\mathcal{C}$ be a committee of size at least $c \log n$ in the i'th functionality round. The probability that $\mathcal{J}_i$ is disjoint of $\mathcal{C}$ is bounded by:

$$\begin{aligned}
\Pr[\mathcal{C} \cap \mathcal{J}_i = \emptyset] &= \prod_{k=0}^{|\mathcal{C}|-1} \left(1 - \frac{|\mathcal{J}_i|}{n-k}\right) \\
&\leq \left(1 - \frac{|\mathcal{J}_i|}{n}\right)^{|\mathcal{C}|} \\
&\leq e^{-\frac{|\mathcal{J}_i||\mathcal{C}|}{n}} \\
&\leq e^{-\beta' c \log n} \\
&= n^{-\beta' c}.
\end{aligned}$$

Substituting c with $\log(m(32m+10))/\beta' \log n$, we get that $\Pr[\mathcal{C} \cap \mathcal{J}_i = \emptyset] \leq 1/(32m+10)m$. As the number of committees in the i'th round of calls is at most m (recall that m is the total running-time of π), by union bound over these committees we conclude that

$$\Pr\left[E^i_{\overline{\mathcal{J}}}\right] \leq \frac{1}{32m+10}.$$

□

The two-party coin-flipping protocol is a variant of the two-party protocol considered in Section 5.2.1. Fix subsets $\mathcal{J}_1, \ldots, \mathcal{J}_s$ whose existence is guaranteed by Claim 5.8, denote $\mathcal{J} = \bigcup_{i \in [s]} \mathcal{J}_i$, and fix an arbitrary $(1-\beta)n$-size subset $\mathcal{S} \subseteq [n] \setminus \mathcal{J}$. Let $\mathsf{extend}(\mathcal{C})$ be defined as follows. If $|\mathcal{C}| \leq c \log n$, then we define $\mathsf{extend}(\mathcal{C})$ to be the committee $\mathcal{C}'$ obtained from $\mathcal{C}$ by padding it with arbitrary (but fixed) parties from $[n] \setminus (\mathcal{S} \cup \mathcal{J})$, such that $|\mathcal{C}' \setminus (\mathcal{S} \cup \mathcal{J})| = c \log n$. Otherwise ($|\mathcal{C}| > c \log n$), set $\mathsf{extend}(\mathcal{C})$ to an arbitrary $(c \log n)$-size subset of $[n] \setminus (\mathcal{S} \cup \mathcal{J})$. The two-party protocol ψ is defined as follows:

Protocol 5.9 (Protocol $\psi = (\mathsf{P}_0, \mathsf{P}_1)$)**.**

1. *Party P_0 controls the parties in $\mathcal{I}_0 = \mathcal{S} \cup \mathcal{J}$ and P_1 controls the parties in $\mathcal{I}_1 = [n] \setminus (\mathcal{S} \cup \mathcal{J})$, in an emulation of π: in each communication round, P_i (for $i \in \{0,1\}$) emulates internally all messages among parties in $\mathcal{I}_i$. Party P_i also sends to P_{1-i} all messages from parties in $\mathcal{I}_i$ to parties in $\mathcal{I}_{1-i}$ and over the broadcast channel.*

2. *For $i = 1, \ldots, s$: let $\mathcal{C}^{(i)}$ be the set of committees in the i'th functionality round of the emulation, and let*

$$\mathcal{C}^{(i)}_{\cap} = \left\{\mathcal{C} \in \mathcal{C}^{(i)} : \mathcal{C} \cap \mathcal{J}_i \neq \emptyset\right\} \quad \text{and} \quad \mathcal{C}^{(i)}_{\not\cap} = \left\{\mathcal{C} \in \mathcal{C}^{(i)} : \mathcal{C} \cap \mathcal{J}_i = \emptyset\right\}.$$

   For each committee $\mathcal{C} \in \mathcal{C}^{(i)}_{\cap}$, the parties assume the call by $\mathcal{C}$ is aborted by the parties in $\mathcal{C} \cap \mathcal{J}_i$. Denote $\mathcal{C}^{(i)}_{\not\cap} = \mathcal{C}_1, \ldots, \mathcal{C}_\ell$; then P_0 sends ℓ separate messages, one for each committee, each containing a uniform bit as the result of the call by that committee.

Output: *Party P_0 (resp., P_1) outputs the output value of an arbitrary non-aborted party in $\mathcal{I}_0$ (resp., $\mathcal{I}_1$) in π.*

Dealing with abort:

1. Party P_i aborts during a communication round: *party P_{1-i} continues the emulation internally, as if all the parties in $\mathcal{I}_i$ aborted in the current round of π. If P_1 continues the emulation internally, it emulates all remaining calls to the ideal functionality by itself.*

2. Party P_0 aborts during a functionality round: *let $\mathcal{C}_1, \ldots, \mathcal{C}_\ell$ be the committees of the current functionality round, that are not aborted by the parties in $\mathcal{J}$, and let $\mathcal{C}_j$ be the committee corresponding to the aborting round. Party P_1 continues the emulation by itself, as if the minimal-index party in $\mathcal{C}_j$ aborted the call to the ideal functionality, and flips the remaining coins for $\mathcal{C}_{j+1}, \ldots, \mathcal{C}_\ell$ internally. Then, P_1 decides on its output as follows: it chooses uniformly at random a $(1-\beta)n$-size subset $\mathcal{T} \subseteq [n] \setminus (\mathcal{S} \cup \mathcal{J} \cup \mathsf{extend}(\mathcal{C}_j))$, and continues the emulation of π internally as if all remaining parties outside $\mathcal{T}$ aborted right after the call to the ideal functionality.*

Note that in the i'th functionality round, party P_0 sends uniform bits only for committees that do not intersect with $\mathcal{J}_i$. For the other committees, the parties assume their calls were aborted by the parties in $\mathcal{J}_i$.

The proof of the theorem proceeds as follows. Similarly to Section 5.2.1, we turn to translate the guaranteed attacker against the two-party protocol ψ into an attacker on π. The attacker $\mathsf{A}^{\pi}_{\mathcal{T}}$ on π, follows closely the attacker from the no-large-committees case, with the following adjustment. To simulate the addition of the aborting subsets to the two-party protocol $\mathsf{A}^{\pi}_{\mathcal{T}}$, which corrupts the pre-designated aborting subsets $\mathcal{J}_1, \ldots, \mathcal{J}_s$, simply instructs each of them to abort in the functionality rounds dictated by ψ. In the case where all calls by large committees are indeed aborted, we are essentially left with the no-large-committees case, and the proof then proceeds as in Section 5.2.1.

Let A^{ψ} be the attacker for ψ guaranteed by Lemma 5.5. As in Section 5.2.1, we assume without loss of generality that A^{ψ} corrupts P_0 and might abort in round i^* of ψ, that corresponds to a functionality round in π. Let $\mathcal{C}^*$ be the committee in π that corresponds to round i^* in ψ, and let

i_f^* be the index of the functionality round in π in which the committee members of $\mathcal{C}^*$ invoke the ideal functionality.

The main difference between the above attacker and the one we used in Section 5.2.1, is that in the above, if the size of the "attacked committee" is too large, i.e., if $|\mathcal{C}^*| > c \log n$, then $\mathcal{C}^*$ is no longer a subset of $\mathsf{extend}(\mathcal{C}^*)$. This seems to be problematic, since the attack on protocol π given in Section 5.2.1, crucially relies on the fact that $\mathcal{C}^* \subseteq \mathsf{extend}(\mathcal{C}^*)$. Fortunately, by the choice of the sets $\mathcal{J}_1, \ldots, \mathcal{J}_s$, the committee $\mathcal{C}^*$ will be large only with very small probability, and thus the resulting attack on π will go through. We formalize this intuition by considering an attacker $\widehat{\mathsf{A}}^\psi$ on protocol ψ that never aborts if the attacked committee is larger than $c \log n$, and still noticeably biases the protocol ψ. Formally, the attacker $\widehat{\mathsf{A}}^\psi$ acts as A^ψ, but with the following change: let $\mathcal{C}^*$ be committee corresponding to round i^* (in which A^ψ is about to abort). If $|\mathcal{C}^*| > c \log n$, then $\widehat{\mathsf{A}}^\psi$ does not abort (even if A^ψ does). Recall that A^ψ might wish to abort the call of $\mathcal{C}^*$ only if $\mathcal{C}^*$ does not intersect $\mathcal{J}_{i_f^*}$ (otherwise, P_0 does not send an answer bit to $\mathcal{C}^*$ in ψ). By Claim 5.8, if $|\mathcal{C}^*| > c \log n$, the latter happens with probability at most $1/(32m + 10)$. Combining this together with Lemma 5.5, it holds that the bias of P_1 in an execution of ψ with $\widehat{\mathsf{A}}^\psi$ is at least $1/(32m + 8)$.

For a $(1 - \beta)n$-size subset $\mathcal{T} \subseteq [n] \setminus (\mathcal{S} \cup \mathcal{J})$, the attacker $\mathsf{A}^\pi_\mathcal{T}$ corrupting $[n] \setminus \mathcal{T}$ is defined as follows:

Algorithm 5.10 (The fail-stop attacker $\mathsf{A}^\pi_\mathcal{T}$ for π)**.**

1. *Start emulating $\widehat{\mathsf{A}}^\psi$, by forwarding to $\widehat{\mathsf{A}}^\psi$ the messages sent by the honest party in ψ and the results of the calls to the ideal functionality.*
2. *For $i = 1, \ldots, i_f^*$: for each committee $\mathcal{C} \in \mathcal{C}^{(i)}$, instruct the parties in $\mathcal{C} \cap \mathcal{J}_i$ to abort the call of $\mathcal{C}$ to the ideal functionality.*
3. *In the i_f^* invocation:*
   (a) *If $\mathsf{extend}(\mathcal{C}^*) \subseteq [n] \setminus \mathcal{T}$: wait until all other committees but $\mathcal{C}^*$ in the i_f^* round of calls are done. Let $\mathcal{C}_1, \ldots, \mathcal{C}_\ell$ be the committees that call the ideal functionality in the i_f^*'th functionality round of π, and correspond to rounds earlier than i^* in ψ. Give the result bits of $\mathcal{C}_1, \ldots, \mathcal{C}_\ell$ to $\widehat{\mathsf{A}}^\psi$ sequentially. If $\widehat{\mathsf{A}}^\psi$ chooses to abort, instruct the minimal-index party in $\mathcal{C}^*$ to abort the call to the ideal functionality, and in the next ordinary communication round of π, instruct all parties in $[n] \setminus \mathcal{T}$ to abort. If $\widehat{\mathsf{A}}^\psi$ does not abort, complete the honest execution of π.*
   (b) *If $\mathsf{extend}(\mathcal{C}^*) \not\subseteq [n] \setminus \mathcal{T}$: continue in the honest execution of π.*
4. *If not aborted after invocation i_f^*, then for $i = i_f^* + 1, \ldots, s$: for each committee $\mathcal{C} \in \mathcal{C}^{(i)}$, instruct the parties in $\mathcal{C} \cap \mathcal{J}_i$ to abort the call of $\mathcal{C}$ to the ideal functionality.*

The success bias obtained by $\mathsf{A}^\pi_\mathcal{T}$ is analyzed in following lemma, which immediately yields the proof of Theorem 5.2.

Lemma 5.11. *There exists a $(1 - \beta)n$-subset $\mathcal{T} \subseteq [n] \setminus (\mathcal{S} \cup \mathcal{J})$ such that $\mathsf{A}^\pi_\mathcal{T}$ biases the output of the honest parties in π by $\Omega(n^{-c}/m)$.*

Proof. The following proof follows similar lines to the proof of Lemma 5.7.

Recall our notation: let i^* be the round in ψ in which $\widehat{\mathsf{A}}^{\psi}$ might abort, let $\mathcal{C}^*$ be the committee in π that corresponds to round i^* in ψ, and let i^*_f be the index of the functionality round in π in which the committee members of $\mathcal{C}^*$ call the ideal functionality. Denote $\widehat{\mathcal{C}^*} = \mathsf{extend}(\mathcal{C}^*)$ and hence, by construction, $|\widehat{\mathcal{C}^*} \setminus (\mathcal{S} \cup \mathcal{J})| = c \log n$. Finally, recall that $\mathcal{T} \subseteq [n] \setminus (\mathcal{S} \cup \mathcal{J})$ and that $\mathsf{A}^{\pi}_{\mathcal{T}}$ corrupts the parties in $[n] \setminus \mathcal{T}$.

Let $B_{\mathcal{T}}$ denote the bias of the output of the honest parties in π, in a random execution of π in which $\mathsf{A}^{\pi}_{\mathcal{T}}$ controls the parties in $[n] \setminus \mathcal{T}$. Let V denote the view of the parties in $\mathcal{S} \cup \mathcal{J}$ up to the i^*_f functionality round, including the answer of the calls made in this round by the committees appearing before $\mathcal{C}^*$ in the (arbitrary) order of ψ. As in the proof of Lemma 5.7, it holds that

$$\begin{aligned} \mathrm{E}_{\mathcal{T}}\left[\mathrm{E}_{v}\left[\mathrm{E}\left[B_{\mathcal{T}} \mid V=v\right]\right]\right] = \mathrm{E}_{v}\left[\mathrm{E}_{\mathcal{T}\mid \mathcal{T}\cap\widehat{\mathcal{C}^*}=\emptyset}\left[\mathrm{E}\left[B_{\mathcal{T}} \mid V=v\right]\right]\right] \cdot \Pr\left[\mathcal{T}\cap\widehat{\mathcal{C}^*}=\emptyset\right] \\ + \mathrm{E}_{v}\left[\mathrm{E}_{\mathcal{T}\mid \mathcal{T}\cap\widehat{\mathcal{C}^*}\neq\emptyset}\left[\mathrm{E}\left[B_{\mathcal{T}} \mid V=v\right]\right]\right] \cdot \Pr\left[\mathcal{T}\cap\widehat{\mathcal{C}^*}\neq\emptyset\right]. \end{aligned} \tag{9}$$

The above expectations are with respect to $v \leftarrow V$ and $\mathcal{T} \leftarrow [n] \setminus (\mathcal{S} \cup \mathcal{J})$ of size $(1-\beta)n$.

As in the proof of Lemma 5.7, it holds that $\mathrm{E}_{v}\left[\mathrm{E}_{\mathcal{T}\mid \mathcal{T}\cap\widehat{\mathcal{C}^*}=\emptyset}\left[\mathrm{E}\left[B_{\mathcal{T}} \mid V=v\right]\right]\right]$ is the bias of the honest party in ψ in an execution with $\widehat{\mathsf{A}}^{\psi}$. Note it might be the case that $\mathcal{T}\cap\widehat{\mathcal{C}^*}=\emptyset$, but $\mathcal{T}\cap\mathcal{C}^*\neq\emptyset$ (if $|\mathcal{C}^*| > c\log n$, and hence $\widehat{\mathcal{C}^*}$ is an arbitrary $(c\log n)$-size subset of $[n]\setminus(\mathcal{S}\cup\mathcal{J})$), so $\mathsf{A}^{\pi}_{\mathcal{T}}$ cannot abort the call by committee $\mathcal{C}^*$. But this is benign, since in case $|\mathcal{C}^*| > c\log n$, $\widehat{\mathsf{A}}^{\psi}$ does not abort. Hence,

$$\mathrm{E}_{v}\left[\mathrm{E}_{\mathcal{T}\mid \mathcal{T}\cap\widehat{\mathcal{C}^*}=\emptyset}\left[\mathrm{E}\left[B_{\mathcal{T}} \mid V=v\right]\right]\right] \geq 1/(32m+8). \tag{10}$$

Similarly, $\mathrm{E}_{v}\left[\mathrm{E}_{\mathcal{T}\mid \mathcal{T}\cap\widehat{\mathcal{C}^*}\neq\emptyset}\left[\mathrm{E}\left[B_{\mathcal{T}} \mid V=v\right]\right]\right]$ is the bias achieved by an attacker that always instructs the parties in $\mathcal{J}_1,\ldots,\mathcal{J}_s$ to abort (subset $\mathcal{J}_i$ in the i'th functionality round), but nothing else. We assume without loss of generality that

$$\mathrm{E}_{v}\left[\mathrm{E}_{\mathcal{T}\mid \mathcal{T}\cap\widehat{\mathcal{C}^*}\neq\emptyset}\left[\mathrm{E}\left[B_{\mathcal{T}} \mid V=v\right]\right]\right] < c' \cdot n^{-c}/m, \tag{11}$$

for $c' > 0$ to be determined later by the analysis, as otherwise, an attacker corrupting $\mathcal{J}_1,\ldots,\mathcal{J}_s$ biases the output of the honest parties in π by $c' \cdot n^{-c}/m$, concluding the lemma proof.

A simple counting argument yields that

$$\Pr\left[\mathcal{T}\cap\widehat{\mathcal{C}^*}=\emptyset\right] \geq \left(\frac{2\beta-1}{\beta e}\right)^{c\log n}. \tag{12}$$

Putting everything together, we conclude that

$$\mathrm{E}_{\mathcal{T}}\left[\mathrm{E}_{v}\left[\mathrm{E}\left[B_{\mathcal{T}} \mid V=v\right]\right]\right] \geq \frac{1}{32m+8}\left(\frac{2\beta-1}{e}\right)^{c\log n} - c' \cdot n^{-c}/m,$$

which is in $\Omega(n^{-c}/m)$ for small enough c'. It follows that there exists a set $\mathcal{T} \in [n]\setminus(\mathcal{S}\cup\mathcal{J})$ such that $\mathsf{A}^{\pi}_{\mathcal{T}}$ biases the output of the honest parties in π by $\Omega(n^{-c}/m)$. □

5.3 A Single-Aborting-Round Variant of Cleve's Attacker

Cleve [21] showed that for every (plain-model) r-round, γ-consistent, two-party coin-flipping protocol, there exists an efficient fail-stop attacker that by corrupting one of the parties, can bias the output of the other party by at least $\gamma/(4r+1)$. In more detail, let $\pi = (\mathsf{P}_0, \mathsf{P}_1)$ be a two-party, r-round, γ-consistent coin-flipping protocol. The attacker A guaranteed by [21] is defined as follows: Assume for concreteness that A is corrupting P_0 and would like to bias the output of P_1 toward one. The attacker A follows the protocol honestly up to some pre-defined round $i^* \in [r]$, in which P_0 is about to send a message. Then, before sending the message, it examines the value of its output in the case that P_1 aborts right after receiving the message; denote this value by b_{i^*}. If $b_{i^*} = 0$, the attacker A instructs P_0 to abort in round i^*; otherwise, A instructs P_0 to honestly send its i^*'th message and abort right after doing that.

The following lemma establishes the existence of an even simpler kind of a fail-stop attacker, one that might abort only in a single pre-defined round.

Lemma 5.12 (Restatement of Lemma 5.5)**.** *Let $\pi = (\mathsf{P}_0, \mathsf{P}_1)$ be a two-party, r-round, γ-consistent coin-flipping protocol. Then, there exist a party $\mathsf{P} \in \{\mathsf{P}_0, \mathsf{P}_1\}$, a round index $i^* \in [r]$, an index $j^* \in \{i^*, i^*-1\}$, and a bit $b \in \{0,1\}$ such that the following holds. Consider the fail-stop attacker A', that corrupting P, instructs P to abort in round i^* if $b_{j^*} = b$. Then, A' biases the output of the other party by $\gamma/(8r+2)$.*

Proof. Let A be the fail-stop attacker for π guaranteed by [21]: A controls party $\mathsf{P} \in \{\mathsf{P}_0, \mathsf{P}_1\}$, and aborts in round j^* if $b_{j^*} = b$ for some $b \in \{0,1\}$; otherwise A aborts in round j^*+1. (The attacker guaranteed by [21] might also instruct P_0 to abort in the very first round; it is easy to see, however, that this attacker also yields an attacker of the type stated in the lemma.) Assume for concreteness that $\mathsf{P} = \mathsf{P}_0$ and $b = 0$. Consider the following two adversaries, both corrupting P_0:

- Attacker A_0 follows the protocol honestly up to round j^*. Then, if $b_{j^*} = 0$, it aborts in round j^*; otherwise, it continues with the execution of the protocol honestly.
- Attacker A_1 follows the protocol honestly up to round j^*+1. Then, if $b_{j^*} = 1$, it aborts in round j^*+1; otherwise, it continues with the execution of the protocol honestly.

We prove that the average of the bias the two adversaries achieve is half the bias of A. Recalling that the bias of A is at least $\gamma/(4r+1)$, this will complete the proof. Let O be a random variable denoting the output of P_1 in $(\mathsf{A}, \mathsf{P}_1)$. Also, denote by Q_j the event in which P_0 aborted in round j, and by $\neg Q$ the event in which P_0 did not abort. The bias achieved by A can be written as:

$$B = \Pr[b_{j^*} = 0] \cdot \mathrm{E}[O \mid b_{j^*} = 0 \wedge Q_{j^*}] + \Pr[b_{j^*} = 1] \cdot \mathrm{E}[O \mid b_{j^*} = 1 \wedge Q_{j^*+1}] - \tfrac{1}{2}. \tag{13}$$

Similarly, we can write the bias of A_0 and of A_1 as:

$$B_0 = \Pr[b_{j^*} = 0] \cdot \mathrm{E}[O \mid b_{j^*} = 0 \wedge Q_{j^*}] + \Pr[b_{j^*} = 1] \cdot \mathrm{E}[O \mid \neg Q] - \tfrac{1}{2}.$$
$$B_1 = \Pr[b_{j^*} = 1] \cdot \mathrm{E}[O \mid b_{j^*} = 0 \wedge Q_{j^*+1}] + \Pr[b_{j^*} = 0] \cdot \mathrm{E}[O \mid \neg Q] - \tfrac{1}{2}.$$

Taking the average of B_0 and B_1 yields that:

$$\begin{aligned}\tfrac{1}{2}(B_0+B_1) &= \tfrac{1}{2}\left(\Pr[b_{j^*}=0]\cdot \mathrm{E}[O \mid b_{j^*}=0 \wedge Q_{j^*}] + \Pr[b_{j^*}=1]\cdot \mathrm{E}[O \mid b_1=0 \wedge Q_{j^*+1}] - \tfrac{1}{2}\right)\\ &+ \tfrac{1}{2}\left(\Pr[b_{j^*}=1]\cdot \mathrm{E}[O \mid \neg Q] + \Pr[b_{j^*}=0]\cdot \mathrm{E}[O \mid \neg Q] - \tfrac{1}{2}\right)\\ &= \tfrac{1}{2}B + \tfrac{1}{2}\left(\mathrm{E}[O \mid \neg Q] - \tfrac{1}{2}\right)\\ &= \tfrac{1}{2}B.\end{aligned}$$

The last equality follows by the fact that $\mathrm{E}[O \mid \neg Q] = \frac{1}{2}$ (the expected output of the honest party in an honest execution of π). □

Bibliography

[1] B. Alon and E. Omri. Almost-optimally fair multiparty coin-tossing with nearly three-quarters malicious. In *Proceedings of the 14th Theory of Cryptography Conference, TCC 2016-B, part I*, pages 307–335, 2016.

[2] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology*, 23(2):281–343, 2010.

[3] B. Awerbuch, M. Blum, B. Chor, S. Goldwasser, and S. Micali. How to implement Bracha's $O(\log n)$ Byzantine agreement algorithm, 1985. Unpublished manuscript.

[4] C. Baum, E. Orsini, and P. Scholl. Efficient secure multiparty computation with identifiable abort. In *Proceedings of the 14th Theory of Cryptography Conference, TCC 2016-B, part I*, pages 461–490, 2016.

[5] A. Beimel, Y. Lindell, E. Omri, and I. Orlov. $1/p$-secure multiparty computation without honest majority and the best of both worlds. In P. Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 277–296. Springer, 2011.

[6] A. Beimel, E. Omri, and I. Orlov. Protocols for multiparty coin toss with a dishonest majority. *Journal of Cryptology*, 28(3):551–600, 2015.

[7] A. Beimel, I. Haitner, N. Makriyannis, and E. Omri. Tighter bounds on multi-party coin flipping via augmented weak martingales and differentially private sampling. In *Proceedings of the 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 838–849, 2018.

[8] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1–10, 1988.

[9] E. Boyle, S. Goldwasser, and Y. Tauman Kalai. Leakage-resilient coin tossing. In *Proceedings of the 25th International Symposium on Distributed Computing (DISC)*, pages 181–196, 2011.

[10] E. Boyle, S. Goldwasser, A. Jain, and Y. Tauman Kalai. Multiparty computation secure against continual memory leakage. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1235–1254, 2012.

[11] E. Boyle, S. Goldwasser, and S. Tessaro. Communication locality in secure multi-party computation - how to run sublinear algorithms in a distributed setting. In *Proceedings of the 10th Theory of Cryptography Conference, TCC 2013*, pages 356–376, 2013.

[12] E. Boyle, K. Chung, and R. Pass. Large-scale secure computation: Multi-party computation for (parallel) RAM programs. In *Advances in Cryptology – CRYPTO 2015, part II*, pages 742–762, 2015.

[13] E. Boyle, R. Cohen, D. Data, and P. Hubáček. Must the communication graph of MPC protocols be an expander? In *Advances in Cryptology – CRYPTO 2018, part III*, pages 243–272, 2018.

[14] G. Bracha. An O(log n) expected rounds randomized Byzantine generals protocol. *Journal of the ACM*, 34(4):910–920, 1987.

[15] N. Braud-Santoni, R. Guerraoui, and F. Huc. Fast Byzantine agreement. In *Proceedings of the 32th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 57–64, 2013.

[16] N. Buchbinder, I. Haitner, N. Levi, and E. Tsfadia. Fair coin flipping: Tighter analysis and the many-party case. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2580–2600, 2017.

[17] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13 (1):143–202, 2000.

[18] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 494–503, 2002.

[19] J. F. Canny and S. Sorkin. Practical large-scale distributed key generation. In *Advances in Cryptology – EUROCRYPT 2004*, pages 138–152, 2004.

[20] A. Cevallos, S. Fehr, R. Ostrovsky, and Y. Rabani. Unconditionally-secure robust secret sharing with compact shares. In *Advances in Cryptology – EUROCRYPT 2012*, pages 195–208, 2012.

[21] R. Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC)*, pages 364–369, 1986.

[22] R. Cohen and Y. Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. *Journal of Cryptology*, 30(4):1157–1186, 2017.

[23] R. Cohen, S. Coretti, J. Garay, and V. Zikas. Round-preserving parallel composition of probabilistic-termination cryptographic protocols. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 37:1–37:15, 2017.

[24] R. Cohen, I. Haitner, E. Omri, and L. Rotem. Characterization of secure multiparty computation without broadcast. *Journal of Cryptology*, 31(2):587–609, 2018.

[25] R. Cohen, I. Haitner, E. Omri, and L. Rotem. From fairness to full security in multiparty computation. In *Proceedings of the 11th International Conference on Security and Cryptography for Networks SCN*, pages 216–234, 2018.

[26] R. Cohen, J. A. Garay, and V. Zikas. Broadcast-optimal two-round MPC. In *Advances in Cryptology – EUROCRYPT 2020, part II*, pages 828–858, 2020.

[27] R. Cramer, I. Damgård, and S. Fehr. On the cost of reconstructing a secret, or VSS with optimal reconstruction phase. In *Advances in Cryptology – CRYPTO 2001*, pages 503–523, 2001.

[28] I. Damgård and Y. Ishai. Scalable secure multiparty computation. In *Advances in Cryptology – CRYPTO 2006*, pages 501–520, 2006.

[29] I. Damgård, Y. Ishai, M. Krøigaard, J. B. Nielsen, and A. D. Smith. Scalable multiparty computation with nearly optimal work and resilience. In *Advances in Cryptology – CRYPTO 2008*, pages 241–261, 2008.

[30] V. Dani, V. King, M. Movahedi, J. Saia, and M. Zamani. Secure multi-party computation in large networks. *Distributed Computing*, 30(3):193–229, 2017.

[31] U. Feige. Noncryptographic selection protocols. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 142–153, 1999.

[32] O. Goldreich. *Foundations of Cryptography – VOLUME 2: Basic Applications*. Cambridge University Press, 2004.

[33] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[34] S. Goldwasser and Y. Lindell. Secure multi-party computation without agreement. *Journal of Cryptology*, 18(3):247–287, 2005.

[35] D. Gordon and J. Katz. Complete fairness in multi-party computation without an honest majority. In *Proceedings of the 6th Theory of Cryptography Conference, TCC 2009*, pages 19–35, 2009.

[36] D. Gordon and J. Katz. Partial fairness in secure two-party computation. In H. Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 157–176. Springer, 2010.

[37] R. Gradwohl, S. P. Vadhan, and D. Zuckerman. Random selection with an adversarial majority. In *Advances in Cryptology – CRYPTO 2006*, pages 409–426, 2006.

[38] I. Haitner and E. Tsfadia. An almost-optimally fair three-party coin-flipping protocol. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 817–836, 2014.

[39] C. Hazay and Y. Lindell. *Efficient Secure Two-Party Protocols - Techniques and Constructions*. Information Security and Cryptography. Springer, 2010.

[40] M. Hirt, U. Maurer, and V. Zikas. MPC vs. SFE : Unconditional and computational security. In *Advances in Cryptology – ASIACRYPT 2008*, pages 1–18, 2008.

[41] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 21–30, 2007.

[42] Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer - efficiently. In *Advances in Cryptology – CRYPTO 2008*, pages 572–591, 2008.

[43] Y. Ishai, J. Katz, E. Kushilevitz, Y. Lindell, and E. Petrank. On achieving the "best of both worlds" in secure multiparty computation. *SIAM Journal on Computing*, 40(1):122–141, 2011.

[44] Y. Ishai, R. Ostrovsky, and H. Seyalioglu. Identifying cheaters without an honest majority. In *Proceedings of the 9th Theory of Cryptography Conference, TCC 2012*, pages 21–38, 2012.

[45] Y. Ishai, R. Ostrovsky, and V. Zikas. Secure multi-party computation with identifiable abort. In *Advances in Cryptology – CRYPTO 2014, part II*, pages 369–386, 2014.

[46] Y. Ishai, E. Kushilevitz, M. Prabhakaran, A. Sahai, and C. Yu. Secure protocol transformations. In *Advances in Cryptology – CRYPTO 2016, part II*, pages 430–458, 2016.

[47] B. M. Kapron, D. Kempe, V. King, J. Saia, and V. Sanwalani. Fast asynchronous Byzantine agreement and leader election with full information. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1038–1047, 2008.

[48] A. Kiayias, H. Zhou, and V. Zikas. Fair and robust multi-party computation using a global transaction ledger. In *Advances in Cryptology – EUROCRYPT 2016, part II*, pages 705–734, 2016.

[49] V. King and J. Saia. From almost everywhere to everywhere: Byzantine agreement with $\tilde{O}(n^{3/2})$ bits. In *Proceedings of the 23th International Symposium on Distributed Computing (DISC)*, pages 464–478, 2009.

[50] V. King and J. Saia. Breaking the $o(n^2)$ bit barrier: scalable Byzantine agreement with an adaptive adversary. In *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 420–429, 2010.

[51] V. King, J. Saia, V. Sanwalani, and E. Vee. Scalable leader election. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 990–999, 2006.

[52] V. King, S. Lonargan, J. Saia, and A. Trehan. Load balanced scalable Byzantine agreement through quorum building, with full information. In *Proceedings of the 12th International Conference on Distributed Computing and Networking (ICDCN)*, pages 203–214, 2011.

[53] Y. Lindell and T. Rabin. Secure two-party computation with fairness - A necessary design principle. In *Proceedings of the 15th Theory of Cryptography Conference, TCC 2017, part I*, pages 565–580, 2017.

[54] N. Makriyannis. On the classification of finite Boolean functions up to fairness. In *Proceedings of the 9th Conference on Security and Cryptography for Networks (SCN)*, volume 8642 of *Lecture Notes in Computer Science*, pages 135–154. Springer, 2014.

[55] T. Moran, M. Naor, and G. Segev. An optimally fair coin toss. In *Proceedings of the 6th Theory of Cryptography Conference, TCC 2009*, pages 1–18, 2009.

[56] M. Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991. Preliminary version in *CRYPTO'89*.

[57] R. Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 232–241, 2004.

[58] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 73–85, 1989.

[59] A. Shamir. How to share a secret. *Journal of the ACM*, 22(11):612–613, 1979.

[60] J. Steiner. Über das größte product der theile oder summanden jeder zahl. *Journal für die reine und angewandte Mathematik*, page 208, 1850.

[61] V. Zikas, S. Hauser, and U. Maurer. Realistic failures in secure multi-party computation. In *Proceedings of the 6th Theory of Cryptography Conference, TCC 2009*, pages 274–293, 2009.

A Fairness with Restricted Identifiable Abort (no Inputs)

We start by introducing a variant of fairness with identifiable abort that will be used as an intermediate step in our constructions. This definition captures the delegation of the computation to smaller committees that independently carry out the (same) fair computation, such that the adversary can only abort the computation of committees with corrupted parties. Looking ahead, in the honest-majority setting we will consider a vector of committees $\boldsymbol{\mathcal{C}} = (\mathcal{C}_1, \ldots, \mathcal{C}_\ell)$ that run in parallel, whereas in the no-honest-majority case we will consider a single committee $\mathcal{C}$ (i.e., $\ell = 1$).

Ideal model with fairness and restricted identifiable abort (no inputs). An ideal computation, with $\boldsymbol{\mathcal{C}}$-identifiable-abort, of a no-input, n-party functionality f for parties $(\mathsf{P}_1, \ldots, \mathsf{P}_n)$ with respect to $\boldsymbol{\mathcal{C}} = (\mathcal{C}_1, \ldots, \mathcal{C}_\ell)$, where $\mathcal{C}_1, \ldots, \mathcal{C}_\ell \subseteq [n]$, in the presence of an ideal-model adversary A controlling the parties indexed by $\mathcal{I} \subseteq [n]$, proceeds via the following steps.

Early abort: A can abort the computation by choosing an index of a corrupted party $i_j^* \in \mathcal{I} \cap \mathcal{C}_j$ for every $j \in [\ell]$ and sending the abort message $(\mathsf{abort}, \{i_1^*, \ldots, i_\ell^*\})$ to the trusted party. In case of such abort, the trusted party sends the message $(\perp, \{i_1^*, \ldots, i_\ell^*\})$ to all parties and halts.

Trusted party answers the parties: If $\mathcal{C}_j \subseteq \mathcal{I}$ for some $j \in [\ell]$, the trusted party receives from the adversary output values $y'_1, \ldots, y'_n$ and sends y'_i to P_i. Otherwise, the trusted party uniformly samples random coins r, computes $(y_1, \ldots, y_n) = f(\lambda, \ldots, \lambda; r)$, and sends y_i to party P_i for every $i \in [n]$.

Outputs: Honest parties always output the message received from the trusted party and the corrupted parties output nothing. The adversary A outputs an arbitrary function of the messages received by the corrupted parties from the trusted party and its auxiliary input.

Definition A.1 (ideal-model computation with fairness and restricted identifiable abort (no inputs))**.** *Let $f\colon (\{0,1\}^*)^n \mapsto (\{0,1\}^*)^n$ be a no-input, n-party functionality, let $\mathcal{I} \subseteq [n]$, and let $\boldsymbol{\mathcal{C}} = (\mathcal{C}_1, \ldots, \mathcal{C}_\ell)$, where $\mathcal{C}_1, \ldots, \mathcal{C}_\ell \subseteq [n]$. The* joint execution of f with $\boldsymbol{\mathcal{C}}$ under (A, I) in the ideal model*, on auxiliary input z to A and security parameter κ, denoted $\mathrm{IDEAL}^{\boldsymbol{\mathcal{C}}\text{-id-fair}}_{f,\mathcal{I},\mathsf{A}(z)}(\kappa)$, is defined as the output vector of $\mathsf{P}_1, \ldots, \mathsf{P}_n$ and $\mathsf{A}(z)$ resulting from the above described ideal process.*

Definition A.2. *Let $f\colon (\{0,1\}^*)^n \mapsto (\{0,1\}^*)^n$ be a no-input, n-party functionality and let π be a probabilistic polynomial-time protocol computing f. The* protocol π (δ, t)-securely computes f with fairness and (ℓ, n', t')-identifiable-abort (and computational security)*, if for every probabilistic polynomial-time real-model adversary A, there exists a probabilistic polynomial-time adversary S for the ideal model, such that for every $\mathcal{I} \subseteq [n]$ of size at most t and subsets $\boldsymbol{\mathcal{C}} = (\mathcal{C}_1, \ldots, \mathcal{C}_\ell)$ satisfying $\mathcal{C}_j \subseteq [n]$, $|\mathcal{C}_j| = n'$, and $|\mathcal{I} \cap \mathcal{C}_j| \leq t'$, for every $j \in [\ell]$, it holds that*

$$\left\{\mathrm{REAL}_{\pi,\mathcal{I},\mathsf{A}(z)}(\kappa)\right\}_{z\in\{0,1\}^*,\kappa\in\mathbb{N}} \equiv_c^\delta \left\{\mathrm{IDEAL}^{\boldsymbol{\mathcal{C}}\text{-id-fair}}_{f,\mathcal{I},\mathsf{S}(z)}(\kappa)\right\}_{z\in\{0,1\}^*,\kappa\in\mathbb{N}}.$$

If δ is negligible, we say that π is a protocol that t-securely computes f with fairness and (ℓ, n', t')-identifiable-abort and computational security. We denote fairness with $(1, n', t')$-identifiable-abort by fairness with (n', t')-identifiable-abort.

The protocol π (δ, t)-securely computes f with fairness and (ℓ, n', t')-identifiable-abort (and information-theoretic (statistical) security)*, if for every real-model adversary A, there exists an adversary S for the ideal model, whose running time is polynomial in the running time of A, such that*

for every $\mathcal{I} \subseteq [n]$ of size at most t, and subsets $\boldsymbol{\mathcal{C}} = (\mathcal{C}_1, \ldots, \mathcal{C}_\ell)$ as above, it holds that

$$\left\{\mathrm{REAL}_{\pi,\mathcal{I},\mathsf{A}(z)}(\kappa)\right\}_{z\in\{0,1\}^*,\kappa\in\mathbb{N}} \equiv_s^{\delta} \left\{\mathrm{IDEAL}^{\boldsymbol{\mathcal{C}}\text{-id-fair}}_{f,\mathcal{I},\mathsf{S}(z)}(\kappa)\right\}_{z\in\{0,1\}^*,\kappa\in\mathbb{N}}.$$

If δ is negligible, we say that π is a protocol that t-securely computes f with fairness and (ℓ, n', t')-identifiable-abort and statistical security.