On Achieving the "Best of Both Worlds" in Secure Multiparty Computation*

Yuval Ishai† Jonathan Katz‡ Eyal Kushilevitz† Yehuda Lindell§
Erez Petrank†

Abstract

Two settings are traditionally considered for secure multiparty computation, depending on whether or not a majority of the parties are assumed to be honest. Protocols designed under this assumption provide "full security" (and, in particular, guarantee output delivery and fairness) when this assumption holds; unfortunately, these protocols are completely insecure if this assumption is violated. On the other hand, protocols tolerating an arbitrary number of corruptions do not guarantee fairness or output delivery even if only a *single* party is dishonest.

It is natural to wonder whether it is possible to achieve the "best of both worlds": namely, a single protocol that simultaneously achieves the best possible security in both the above settings. Here, we rule out this possibility (at least for general functionalities) but show some positive results regarding what *can* be achieved.

Keywords: Theory of cryptography, secure computation.

---

*This is the full version of [18, 19]. This research was supported by grant 36/03 from the Israel Science Foundation, US-Israel Binational Science Foundation grant #2004240, and NSF CAREER award #0447075

†Department of Computer Science, Technion, Israel. Email: {yuvali,eyalk,erez}@cs.technion.ac.il

‡Department of Computer Science, University of Maryland, USA. Email: jkatz@cs.umd.edu. Portions of this work were done while the author was visiting the Institute for Pure and Applied Mathematics (IPAM), UCLA.

§Department of Computer Science, Bar-Ilan University, Israel. Email: lindell@cs.biu.ac.il. Portions of this work were done while the author was visiting the Technion.

Contents

1 Introduction

Protocols for secure multiparty computation (MPC) [22, 11, 4, 8] allow a set of mutually distrusting parties to compute a function in a distributed fashion while guaranteeing (to the extent possible) the privacy of the parties' inputs and the correctness of their outputs. Security is typically formulated by requiring that a real execution of a protocol be indistinguishable from an ideal execution in which the parties hand their inputs to a trusted party who computes the function and returns the outputs to the appropriate parties. Thus, whatever security is implied by the ideal model must also be guaranteed in a real-world execution of the protocol.

The vast body of research in this area can be divided into two, almost disjoint, lines of work: one dealing with the case when a majority of the parties are assumed to be honest, and the other dealing with the case when an arbitrary number of parties may be corrupted. These settings differ not only in the approaches that are used to construct secure protocols, but also in the results that can be achieved (and hence in the ideal models thus defined). In further detail:[1]

- **Secure computation with an honest majority.** When a majority of the participants are honest, it is possible to obtain the strongest level of security one could hope for (i.e., "full security") [11]. Fully secure protocols ensure not only privacy and correctness but also *fairness* (namely, if one party receives its output then all parties do) and *guaranteed output delivery* (i.e., honest parties are guaranteed to successfully complete the computation). In the presence of an honest majority, full security can even be obtained *unconditionally* [4, 8, 2, 20, 10].

- **Secure computation with no honest majority.** Results of Cleve [9] imply that full security (for general functionalities) is *only* possible when an honest majority is present. Specifically, while privacy and correctness are still attainable without an honest majority, it is impossible (in general) to guarantee fairness or output delivery. Thus, when no honest majority is assumed (which includes two-party computation as a special case) a relaxed notion of security ("security with abort") is used where privacy and correctness still hold but the adversary is allowed to *abort* the computation after obtaining its own outputs. Unconditional security is no longer possible in this setting (for general functionalities), but protocols realizing this notion of security for any number of corrupted parties can be constructed based on the existence of enhanced trapdoor permutations or oblivious transfer [22, 11, 3, 13, 14].

An unfortunate drawback of existing protocols for each of the above settings is that they do not provide any security beyond what is implied by the definitions. Specifically, existing protocols designed for the case of honest majority are *completely insecure* once half (or more) of the parties are corrupted: e.g., honest parties' inputs are entirely revealed, and even correctness may be violated. On the other hand, known protocols that achieve security with abort against an arbitrary number of corruptions do not guarantee fairness or output delivery even if only a *single* party is corrupted.

To get a sense for the magnitude and importance of this problem, consider trying to decide which type of protocol is more appropriate to implement secure voting. Since we would like privacy of individuals' votes to hold regardless of the number of corruptions, we are forced to use a protocol of the second type that provides only security with abort. But then a single corrupted machine (in fact, even one which simply fails in the middle of the election) can perform a denial-of-service attack that prevents all honest parties from learning the outcome. Neither option is very appealing.

[1] Here and in the rest of the paper, we refer to the standard model of MPC over a synchronous network of secure point-to-point channels and broadcast channels; see Section 2 for definitions.

The above state of affairs raises the following natural question:

> *To what extent can we design a* **single** *protocol achieving the "best of both worlds" regardless of the number of corruptions; i.e., a protocol that simultaneously guarantees full security in case a majority of the parties are honest, and security with abort otherwise?*

To the best of our knowledge, the above fundamental question has not been studied previously. This constitutes a major gap in our current understanding of the feasibility of secure computation.

1.1 Our Results

In this paper, we initiate the study of the above question. Our main results settle the question in the negative, showing that (in general) it is impossible to construct a single protocol that achieves the best possible security regardless of the number of corruptions. The strength of these negative results depends on whether we consider standard functionalities, which receive n inputs and deliver n outputs in one shot, or *reactive* functionalities which can receive inputs and deliver outputs in multiple phases and maintain a secret state information between phases. (See Section 2 for definitions of these and other standard terms we use.) Specifically, we show:

Theorem 1.1 *Let $t + s \geq n$ with $1 \leq t \leq s$. Then there exists a standard (non-reactive) functionality f for which there is no n-party protocol computing f that is simultaneously (1)* fully secure *when t parties are corrupted, and (2)* secure with abort *when s parties are corrupted.*

In fact, our negative results are even stronger than indicated above. Fix s, t, n as above. For non-reactive functionalities, it is impossible (in general) to simultaneously achieve full security against t *fail-stop* corruptions and *privacy* against s *fail-stop* corruptions. For *reactive* functionalities we show an even stronger result: one cannot (in general) simultaneously obtain full security against t *fail-stop* corruptions and *privacy* against s *semi-honest* corruptions.

In light of the above, we are led to explore what security guarantees *can* be achieved with regard to different corruption thresholds. Considering the natural relaxations, we show two incomparable positive results in this regard. First, we show that when $t + s < n$ (and $t < n/2$) then the "best of both worlds" is, indeed, possible. That is:

Theorem 1.2 *Assume the existence of enhanced trapdoor permutations, and let $t + s < n$ and $t < n/2$. Then for any (reactive or non-reactive) functionality f, there exists a protocol computing f that is simultaneously (1)* fully secure *when t parties are corrupted, and (2)* secure with abort *when s parties are corrupted.*

We also show that in the case of non-reactive functionalities, the optimal security thresholds are obtainable if we restrict to *semi-honest* adversaries when there is no honest majority:

Theorem 1.3 *Assume the existence of enhanced trapdoor permutations, and let $t < n/2$ and $s < n$. Then for any non-reactive functionality f there exists a protocol computing f that is simultaneously (1)* fully secure *against t malicious corruptions, and (2)* fully secure *against s semi-honest corruptions.*

Table 1 summarizes the main corollaries of our results along with pointers to the corresponding theorems from the technical sections. As we discuss next, investigating other approaches remains an interesting open problem.

	Standard functionalities	Reactive functionalities
Privacy against $s < n$ *semi-honest* parties and security against $t < n/2$ malicious parties	Yes Theorem 4.2	No Theorem 3.1
Privacy against $s < n$ *malicious* parties and security against $t < n/2$ malicious parties	No Theorem 3.5	No Theorem 3.1

Table 1: Corollaries of our results: the existence and non-existence of protocols that simultaneously guarantee security against $t < n/2$ malicious parties and privacy against $s < n$ malicious or semi-honest parties.

1.2 Future Directions

The stark negative results in this paper, coupled with the fact that it can be difficult to determine what security bounds are appropriate to assume in practice, suggest looking for other ways to obtain "best of both worlds"-type results besides those already discussed in the previous section. We briefly mention two possibilities that have been explored previously, and then highlight some promising directions for future work.

Two interesting feasibility results have been investigated in prior work. Ishai et al. [18] show a protocol for any non-reactive functionality f that provides full security against $t < n/2$ malicious parties, and also ensures the following guarantee against $s < n$ malicious parties (informally): the malicious parties achieve (and learn) no more than they could achieve in s invocations of an ideal party evaluating f, where they may use different inputs in different invocations. For certain functionalities (with voting serving as a prime example), this provides a meaningful notion of security. In another direction, Katz [19] (following [18]) explored what is possible in the case of a *non-rushing* adversary or, equivalently, under the assumption of simultaneous message transmission. Katz shows, for any non-reactive functionality f and any polynomial p, a protocol that is fully secure against $t < n/2$ malicious parties, as well as "$\frac{1}{p}$-secure with abort" for any number of malicious corruptions. Roughly speaking, this latter notion means that the actions of any real-world adversary can be simulated by an ideal-world adversary (who has the ability to abort the protocol) such that the resulting outcomes cannot be distinguished with probability advantage better than $O(1/p)$. (The protocol provides additional security guarantees as well; we refer the reader to [19] for details.)

With regard to future work in this area, several directions seem promising:

- **Non-rushing adversaries.** We currently have only a partial answer to what can be achieved if a non-rushing adversary is assumed. The results of Katz [19] leave open the possibility of protocols in this model that achieve the true "best of both worlds": simultaneous full security against $t < n/2$ corruptions, and security with abort against $s < n$ corruptions. (However, it is known that there are no constant-round [18] or even logarithmic-round [19] protocols of this sort.) Alternatively, it might be possible in this model to obtain full security against $t < n/2$ corruptions and $\frac{1}{p}$-security *without* abort against $s < n$ corruptions. (In fact, this may be possible even for rushing adversaries by building on the ideas of [15].)

- **Specific functionalities.** The impossibility results presented here rule out protocols for *general* functionalities, but leave open the question of what might be obtained for *specific* functionalities of interest. Recent work [16, 17] has shown that protocols with full security *against an arbitrary number of corruptions* can be constructed for certain (non-trivial) functionalities. (This is even better than what a "best of both worlds"-type result would imply.) For what other functionalities might positive results be obtained?

- **Definitional relaxations.** In the current work we have focused on the standard notions of full security and security with abort. Given the impossibility results we have obtained, it may be worthwhile to explore relaxations of these definitions such as those considered in [1, 15].

2 Preliminaries and Definitions

In this work, k denotes the security parameter, and PPT stands for "probabilistic polynomial time".

2.1 Overview

Our default network model consists of n parties, $P_1, \ldots, P_n$, who interact in synchronous rounds via private and authenticated point-to-point channels. We also assume that the parties have access to a broadcast channel. We consider both *rushing* and *non-rushing* adversaries. A rushing adversary may delay sending the messages of the corrupted parties in any given round until *after* the honest parties send their messages in that round; thus, the round-i messages of the corrupted parties may depend on the round-i messages of the honest parties. In contrast, a non-rushing adversary must decide on what messages the corrupted parties should send in any given round *before* seeing the honest parties' messages in that round. Assuming a non-rushing adversary is essentially equivalent to assuming that there exists a mechanism for simultaneous message exchange. The standard definition of secure computation assumes a rushing adversary. Nevertheless, in one of our lower bounds we consider non-rushing adversaries since this only strengthens the result.

We consider both *malicious* adversaries, who have total control over the behavior of corrupted parties and may instruct them to deviate arbitrarily from the protocol specification, and *semi-honest* adversaries who record all information viewed by corrupted parties as they run the protocol but do not otherwise modify their behavior. We also consider *fail-stop* adversaries who follow the protocol honestly (as semi-honest adversaries do) except that they may abort the protocol early.

Throughout the paper, we consider security against computationally bounded adversaries and assume for simplicity that the adversary is *static*, i.e., that the set of corrupted parties is chosen at the onset of the protocol in a non-adaptive manner. This strengthens our negative results, and is not essential for our positive results; see Remark 4.5.

The security of a multi-party protocol is defined with respect to a *functionality* f. A non-reactive n-party functionality is a (possibly randomized) mapping of n inputs to n outputs. A multi-party protocol for computing a non-reactive functionality f is a protocol running in polynomial time and satisfying the following correctness requirement: if parties $P_1, \ldots, P_n$ holding inputs $(1^k, x_i)$, respectively, all run an honest execution of the protocol, then the joint distribution of the outputs $y_1, \ldots, y_n$ of the parties is statistically close to $f(x_1, \ldots, x_n)$.

A *reactive* functionality f is a sequence of non-reactive functionalities $f = (f_1, \ldots, f_\ell)$ computed in a stateful fashion in a series of phases. Let x_i^j denote the input of P_i in phase j, and let s^j denote the state of the computation after phase j. Computation of f proceeds by setting s^0 equal to the empty string and then computing $(y_1^j, \ldots, y_n^j, s^j) \leftarrow f_j(s^{j-1}, x_1^j, \ldots, x_n^j)$ for $j = 1$ to ℓ, where y_i^j denotes the output of P_i at the end of phase j. A multi-party protocol computing f also runs in ℓ phases, at the beginning of which each party holds an input and at the end of which each party obtains an output. (Note that parties may wait to decide on their phase-j input until the beginning of that phase.) Parties maintain state throughout the entire execution. The correctness requirement is that, in an honest execution of the protocol, the joint distribution of all the outputs $\{y_1^j, \ldots, y_n^j\}_{j=1}^{\ell}$ of all the phases is statistically close to the joint distribution of all the outputs of all the phases in a computation of f on the same inputs used by the parties.

2.2 Defining Security

In this section, we present the (standard) security definitions used in this paper. We assume the reader is familiar with the simulation-based approach for defining secure computation, as described in detail in [5, 12, 7]. This definitional approach compares the *real-world* execution of a protocol for computing some function with an *ideal-world* evaluation of the function by a trusted party. Security is then defined by requiring that whatever can be achieved in the real world could have also been achieved (or *simulated*) in the ideal world. More formally, it is required that for every adversary $\mathcal{A}$ attacking the real execution of the protocol there exists an adversary $\mathcal{A}'$, sometimes referred to as a *simulator*, which "achieves the same effect" in the ideal world. This is made more precise in what follows.

The real model. Let π be a multi-party protocol computing a non-reactive functionality f. It is convenient to view an execution of π in the presence of an adversary $\mathcal{A}$ as being coordinated by a non-uniform *environment* $Z = \{Z_k\}$. At the outset, the environment gives input $(1^k, x_i)$ to each party P_i, and gives I, $\{x_i\}_{i \in I}$, and z to $\mathcal{A}$, where $I \subset [n]$ represents the set of corrupted parties and z denotes an auxiliary input. The parties then interact, with each honest (i.e., uncorrupted) party P_i behaving as instructed by the protocol (using input x_i) and corrupted parties behaving as directed by the adversary. In the case of a semi-honest adversary, $\mathcal{A}$ directs the parties to follow the protocol on their given inputs. At the conclusion of the protocol, $\mathcal{A}$ gives to Z an output which is an arbitrary function of $\mathcal{A}$'s view throughout the protocol, and Z is additionally given the outputs of the honest parties. Finally, Z outputs a bit. We let $\textsc{real}_{\pi,\mathcal{A},Z}(k)$ be a random variable denoting the value of this bit.

For reactive functionalities, the environment Z operates in a series of phases. At the outset of the execution, Z gives I and z to $\mathcal{A}$. Then, at the beginning of each phase j, the environment gives input x_i^j to each party P_i and gives $\{x_i^j\}_{i \in I}$ to $\mathcal{A}$. The parties then run the jth phase of protocol π. At the end of each phase, $\mathcal{A}$ gives to Z an output which is an arbitrary function of $\mathcal{A}$'s view thus far, and Z is additionally given the outputs of the honest parties in this phase. If the adversary aborts the protocol in some phase (formally, if the output of some honest party at the end of the phase is $\perp$), execution is halted; otherwise, execution continues until all phases are completed (i.e., the protocol is finished). Once the execution terminates, Z outputs a bit; we let $\textsc{real}_{\pi,\mathcal{A},Z}(k)$ be a random variable denoting the value of this bit.

The ideal model – full security. In the ideal model, there is a trusted party who computes f on behalf of the parties. The first variant of the ideal model, discussed now, corresponds to a notion of security where fairness and output delivery are guaranteed.

Once again, we have an environment Z which provides inputs $x_1, \ldots, x_n$ to the parties, and provides I, $\{x_i\}_{i \in I}$, and z to $\mathcal{A}'$. Execution then proceeds as follows:

- Honest parties send their input x_i to the trusted party. Corrupted parties may send the trusted party arbitrary inputs as instructed by $\mathcal{A}'$. (Any missing or "invalid" value is substituted by a default value.) Denote by x_i' the value sent by party P_i. In the case of a semi-honest adversary, we require that $x_i' = x_i$ for all $i \in I$.

- The trusted party computes $f(x_1', \ldots, x_n') = (y_1, \ldots, y_n)$ and sends y_i to party P_i. (If f is randomized, this computation involves random coins that are generated by the trusted party.)

After the above, $\mathcal{A}'$ gives to Z an output which is an arbitrary function of the view of $\mathcal{A}'$, and Z is also given the outputs of the honest parties. Finally, Z outputs a bit. We let $\text{IDEAL}_{f,\mathcal{A}',Z}(k)$ be the random variable denoting the value of this bit.

For the case of reactive functionalities, execution once again proceeds in a series of phases. At the outset, Z gives I and z to $\mathcal{A}'$. At the beginning of each phase j, the environment provides input x_i^j to party P_i and gives $\{x_i^j\}_{i \in I}$ to $\mathcal{A}'$. Inputs/outputs are then sent to/from the trusted party as above. At the end of each phase, $\mathcal{A}'$ gives to Z an output which is an arbitrary function of its view thus far, and Z is additionally given the outputs of the honest parties in this phase. After all phases have been completed, Z outputs a bit. Once again, we let $\text{REAL}_{\pi,\mathcal{A},Z}(k)$ be a random variable denoting the value of this bit.

The ideal model – security with abort. In this second variant of the ideal model, fairness and output delivery are no longer guaranteed. This is the standard relaxation used when a strict majority of honest parties is not assumed. (Other variants are also possible [12, 14].)

As in the first ideal model, we have an environment Z who provides inputs $x_1, \ldots, x_n$ to the parties, and provides I, $\{x_i\}_{i \in I}$, and z to the adversary $\mathcal{A}'$. Execution then proceeds as follows:

- As before, the parties send their inputs to the trusted party and we let x_i' denote the value sent by P_i. Once again, for a semi-honest adversary we require $x_i' = x_i$ for all $i \in I$.

- The trusted party computes $f(x_1', \ldots, x_n') = (y_1, \ldots, y_n)$ and sends $\{y_i\}_{i \in I}$ to the adversary.

- The adversary chooses whether to continue or abort; this can be formalized by having the adversary send either a continue or abort message to the trusted party. (A semi-honest adversary never aborts.) In the former case, the trusted party sends to each uncorrupted party P_i its output value y_i. In the latter case, the trusted party sends the special symbol $\perp$ to each uncorrupted party.

After the above, $\mathcal{A}'$ gives to Z an output which is an arbitrary function of the view of $\mathcal{A}'$, and Z is also given the outputs of the honest parties. Finally, Z outputs a bit. We let $\text{IDEAL}_{f_\perp,\mathcal{A}',Z}(k)$ be the random variable denoting the value of this bit; the subscript "$\perp$" indicates that the adversary now has the ability to abort the trusted party in the ideal model.

The extension to the case of reactive functionalities is the same as before. As in the real-world model, execution is halted immediately after any phase in which an honest party outputs $\perp$.

Defining security. With the above in place, we can now define our notions of security.

Definition 2.1 (Security, security with abort) *Let π be a multi-party protocol for computing a functionality f, and fix $s \in \{1, \ldots, n\}$.*

1. *We say that π* securely computes f in the presence of malicious (resp., semi-honest) adversaries corrupting s parties *if for any* PPT *adversary (resp., semi-honest adversary) $\mathcal{A}$ there exists a* PPT *adversary (resp., semi-honest adversary) $\mathcal{A}'$ such that for every polynomial-size circuit family $Z = \{Z_k\}$ corrupting at most s parties the following is negligible:*

$$\left|\Pr[\text{REAL}_{\pi,\mathcal{A},Z}(k) = 1] - \Pr[\text{IDEAL}_{f,\mathcal{A}',Z}(k) = 1]\right|.$$

2. *We say that π* securely computes f with abort in the presence of malicious adversaries corrupting s parties *if for any* PPT *adversary $\mathcal{A}$ there exists a* PPT *adversary $\mathcal{A}'$ such that for every polynomial-size circuit family $Z = \{Z_k\}$ corrupting at most s parties the following is negligible:*

$$\left|\Pr[\text{REAL}_{\pi,\mathcal{A},Z}(k) = 1] - \Pr[\text{IDEAL}_{f_\perp,\mathcal{A}',Z}(k) = 1]\right|.$$

We also consider the weaker notion of *privacy* which, roughly speaking, ensures only that the adversary cannot learn anything about honest parties' inputs other than what is implied by its own inputs and outputs. Accordingly, the definition of privacy only requires that the adversary's view in the real model can be simulated in the ideal model.

Formally, define $\text{REAL}'_{\pi,\mathcal{A},Z}(k)$ analogously to $\text{REAL}_{\pi,\mathcal{A},Z}(k)$ except that Z is *not* given the outputs of the honest parties (that is, Z is only given the output of $\mathcal{A}$, which is an arbitrary function of $\mathcal{A}$'s view). We can define $\text{IDEAL}'_{f,\mathcal{A}',Z}(k)$ similarly.

Definition 2.2 (Privacy) *Let π be a multi-party protocol for computing a functionality f, and fix $s \in \{1, \ldots, n\}$. We say that π* privately computes f in the presence of malicious (resp., semi-honest) adversaries corrupting s parties *if for any* PPT *adversary (resp., semi-honest adversary) $\mathcal{A}$ there exists a* PPT *adversary (resp., semi-honest adversary) $\mathcal{A}'$ such that for every polynomial-size circuit family $Z = \{Z_k\}$ corrupting at most s parties the following is negligible:*

$$\left|\Pr[\text{REAL}'_{\pi,\mathcal{A},Z}(k) = 1] - \Pr[\text{IDEAL}'_{f,\mathcal{A}',Z}(k) = 1]\right|.$$

Note that privacy is implied by security with abort.

3 Impossibility Results

3.1 Reactive Functionalities

In this section we present a strong impossibility result for the case of reactive functionalities. The threshold in the theorem that follows is tight; see Section 4.1. The restriction of the theorem to reactive functionalities is also essential, as we show in Section 4.2 a positive result for the case of non-reactive functionalities.

Theorem 3.1 *Let n, t, s be such that $t + s = n$ and $t \geq 1$. There exists a reactive n-party functionality f for which there is no protocol that simultaneously:*

- *securely computes f in the presence of malicious adversaries corrupting t parties;*
- *privately computes f in the presence of semi-honest adversaries corrupting s parties.*

This holds even if the adversary in the first case is restricted to be a non-rushing, fail-stop adversary.

Functionality f

Phase 1:

- *Input:* Party P_1 provides an input bit b (no other party has input).
- *Output:* The functionality records this value but gives no output to any party.

Phase 2:

- *Input:* Party P_1 provides an input bit b' (no other party has input).
- *Output:* P_n outputs b.[a]

[a]Note that b' is not used in the definition of f. We use b' only for the purposes of showing that the protocol is insecure.

Figure 1: The reactive functionality for the proof of Theorem 3.1.

Proof: If $t \geq n/2$ then the theorem follows from the fact that there exist non-reactive functionalities that cannot be computed securely without an honest majority [9]. Thus, we assume $t < n/2$ (implying $n \geq 3$ and $s > 0$) in what follows.

We prove Theorem 3.1 using a two-phase functionality f that corresponds (roughly) to having P_1 commit to a value; see Figure 1. Take any protocol π computing f, and let $S_1 = \{P_1, \ldots, P_t\}$ and $S_2 = \{P_{t+1}, \ldots, P_n\}$; note that $|S_2| = s$. The intuition for the theorem is as follows:

1. Say π privately computes f in the presence of semi-honest adversaries corrupting s parties. Then the parties in S_2 should (jointly) know nothing about b after phase 1.
2. Say π securely computes f in the presence of malicious adversaries corrupting t parties. Then the parties in S_1 should not be able to prevent P_n from learning b in the second phase. In particular, this should hold even if all parties in S_1 abort before the second phase.

Intuitively this gives a contradiction since, following the first phase of the protocol, the parties in S_2 can jointly simulate an execution of the second phase of the protocol when all parties in S_1 abort. The formal proof of the theorem is slightly more involved since item (2), above, is not quite true in the presence of a malicious P_1 who might change his input before running the protocol.

We now prove this formally. Let π be a protocol computing f that is secure in the presence of t non-rushing, fail-stop adversaries; we will show that π cannot also be private in the presence of s semi-honest adversaries. Since π is secure in the presence of t fail-stop adversaries, we may assume without loss of generality that the output of P_n in π is always a bit (and never $\perp$) as long as parties in S_1 behave honestly but may abort the protocol early. We consider two real-world executions of π. In both cases, Z chooses b and b' uniformly and independently.[2]

First execution. Here we consider a non-rushing, fail-stop adversary $\mathcal{A}_1$ who corrupts the parties in S_1 and instructs them to behave as follows: Run the first phase of π honestly. In the second phase, if $b' = b$ then run the second phase honestly; otherwise, abort immediately. We let $\Pr_1[\cdot]$ denote the probability of events in this execution.

Second execution. Here we consider a semi-honest adversary $\mathcal{A}_2$ who corrupts the parties in S_2. At the conclusion of phase 1 of π, this adversary simulates an execution of the second phase of π

[2]Allowing Z to be randomized does not affect our definitions of security.

assuming all parties in S_1 abort, and outputs the resulting output of P_n. We let $\Pr_2[\cdot]$ denote the probability of events in this execution.

Claim 3.2 $\Pr_1[P_n$ *outputs* $b'] = \frac{1}{2} + \frac{1}{2} \cdot (1 - \Pr_2[\mathcal{A}_2$ *outputs* $b])$.

Proof: In the first execution, we can consider a mental experiment in which the parties in S_2 simulate the second phase of π assuming all parties in S_1 abort. By definition of $\mathcal{A}_2$, the probability that the output of P_n in this mental experiment is not equal to b is exactly $1 - \Pr_2[\mathcal{A}_2 \text{ outputs } b]$. Note further that this probability is independent of the value b' used in the second phase.

In the first execution, the parties in S_1 abort with probability exactly 1/2; moreover, when they do not abort the output of P_n is $b = b'$. When the parties in S_1 do abort then P_n outputs b' iff the output of P_n is not equal to b. Thus,

$$\begin{aligned}\Pr_1[P_n \text{ outputs } b'] &= \tfrac{1}{2} + \tfrac{1}{2} \cdot \Pr_1[P_n \text{ outputs } \bar{b} \mid \text{parties in } S_1 \text{ abort}] \\ &= \tfrac{1}{2} + \tfrac{1}{2} \cdot (1 - \Pr_2[\mathcal{A}_2 \text{ outputs } b]),\end{aligned}$$

as desired. ∎

Claim 3.3 $\left|\Pr_1[P_n \textit{ outputs } b'] - \frac{1}{2}\right|$ *is negligible.*

Proof: Here we rely on the assumption that π securely computes f in the presence of a non-rushing, fail-stop adversary corrupting t parties. Consider an ideal-world execution of f in the presence of an adversary corrupting the parties in S_1. In the ideal world, the final output of P_n is equal to whatever value P_1 sends to the trusted party in the first phase. Since the adversary has no information about b' in the first phase, the probability that P_1 sends b' to the trusted party in the first phase (and hence the probability that P_n outputs b' in the second phase) is exactly 1/2. The claim follows. ∎

Claim 3.4 *If* π *privately computes* f *in the presence of a semi-honest adversary corrupting* s *parties, then* $\left|\Pr_2[\mathcal{A}_2 \textit{ outputs } b] - \frac{1}{2}\right|$ *is negligible.*

Proof: This follows from the fact that, in an ideal-world execution of f, any adversary corrupting the parties in S_2 outputs the value of b after phase 1 with probability exactly 1/2. ∎

The preceding three claims imply that π cannot privately compute f in the presence of a semi-honest adversary corrupting s parties. This concludes the proof of Theorem 3.1. ∎

3.2 Non-Reactive Functionalities

We now show an impossibility result for the case of standard (i.e., non-reactive) functionalities. This result is incomparable to the result proved in the previous section, since here we rule out privacy against *malicious* adversaries and explicitly make use of the fact that the adversary can be *rushing*.[3] The thresholds in the theorem are tight; see Section 4.2.

Theorem 3.5 *Let* n, t, s *be such that* $t + s = n$ *and* $t \geq 1$. *Then there exists a non-reactive functionality* $\tilde{f}$ *for which there is no protocol that simultaneously:*

[3] For non-rushing adversaries, similar negative results appear in [18, 19]. However, these apply only to protocols with a small number of rounds (at most logarithmic in the security parameter).

- *securely computes $\tilde{f}$ in the presence of malicious adversaries corrupting t parties;*
- *privately computes $\tilde{f}$ in the presence of malicious adversaries corrupting s parties.*

This holds even if we consider only fail-stop adversaries in each case.

When $t \geq n/2$ the theorem follows from the existence of functionalities that cannot be computed securely without an honest majority [9]. Thus, we assume $t < n/2$ (and hence $n \geq 3$ and $s > 0$) in what follows. We prove the theorem in two stages: In Section 3.2.1 we present a functionality f for which there is no protocol that simultaneously

- securely computes f in the presence of malicious adversaries corrupting t parties;
- securely computes f with abort in the presence of malicious adversaries corrupting s parties.

Extending this, we then show in Section 3.2.2 a (slightly different) functionality $\tilde{f}$ that suffices to prove the theorem.

3.2.1 Ruling out Security with Abort

Fix n, s, and $1 \leq t < n/2$ with $t + s = n$. Define f as follows: parties P_1 and P_n have as input bits b_1 and b_n, respectively, and each receive as output $b_1 \oplus b_n$ (no other parties receive output). Let π be a protocol that securely computes f in the presence of a fail-stop adversary corrupting t parties. We assume that π operates in a fixed number of *segments*, each exactly n rounds long, where only party P_i sends a message in the i^{th} round of a segment. (I.e., in any given segment first P_1 speaks, then P_2, etc. until P_n speaks and then the next segment begins.) If π is secure against a rushing adversary then it can always be transformed into a protocol of this form. Let $r = r(k)$ denote the number of segments of the protocol. We assume that if π is run honestly, then the outputs of P_1 and P_n are correct (and, in particular, agree) with probability[4] at least 7/8.

Define $A \stackrel{\text{def}}{=} \{P_1, \ldots, P_t\}$, $B \stackrel{\text{def}}{=} \{P_{t+1}, \ldots, P_{n-t}\}$, and $C \stackrel{\text{def}}{=} \{P_{n-t+1}, \ldots, P_n\}$. Consider the real-world execution in which Z chooses inputs for P_1 and P_n uniformly and independently (see footnote 2), and then all parties run the protocol honestly except that parties in A or C may (possibly) abort at some round. (Parties in B run the protocol honestly and never abort.) Let v_1^i, with $0 \leq i \leq r$, denote the final output of P_1 when parties in C all abort in segment $i+1$ or, in other words, when segment i is the last segment in which parties in C send any messages. (For $i = 0$ this means that parties in C abort the protocol immediately without sending any messages; for $i = r$ this means that parties in C never abort.) Define v_n^i similarly to be the output of P_n when all parties in A abort in segment $i+1$ (i.e., send messages for the final time in segment i). Note that v_1^i can be computed from the joint view of the parties in $A \cup B$ as soon as all parties in C have sent their segment-i messages, and similarly v_n^i can be computed from the joint view of the parties in $B \cup C$ once all parties in A have sent their segment-i messages.

Security of π implies that, for all i, we have $v_1^i, v_n^i \in \{0,1\}$ (and, in particular, $v_1^i \neq \perp$) with all but negligible probability. This is true since π provides full security against t fail-stop adversaries, and at most t parties abort in the experiment defining v_1^i, v_n^i. In what follows, we will assume for simplicity that $v_1^i, v_n^i \in \{0,1\}$ with probability 1.

[4]Security (or even just correctness) of π actually implies that this holds with all but negligible probability. However, we will make use of the relaxed requirement stated here in Remark 3.6.

Consider the following summation, where all probabilities are with respect to the real-world execution described earlier:

$$\left(\Pr\left[v_1^0 = 1 \wedge v_n^0 = 1\right] + \Pr\left[v_1^0 = 0 \wedge v_n^r = 1\right] - \frac{1}{2}\right) \tag{1}$$

$$+ \quad \left(\Pr\left[v_1^0 = 0 \wedge v_n^0 = 0\right] + \Pr\left[v_1^0 = 1 \wedge v_n^r = 0\right] - \frac{1}{2}\right) \tag{2}$$

$$+ \quad \sum_{i=0}^{r-1} \left[\left(\Pr\left[v_1^i = 0 \wedge v_n^{i+1} = 0\right] + \Pr\left[v_1^i = 1 \wedge v_n^i = 0\right] - \frac{1}{2}\right)\right. \tag{3}$$

$$+ \left(\Pr\left[v_1^i = 1 \wedge v_n^{i+1} = 1\right] + \Pr\left[v_1^i = 0 \wedge v_n^i = 1\right] - \frac{1}{2}\right) \tag{4}$$

$$+ \left(\Pr\left[v_1^{i+1} = 0 \wedge v_n^{i+1} = 0\right] + \Pr\left[v_1^i = 0 \wedge v_n^{i+1} = 1\right] - \frac{1}{2}\right) \tag{5}$$

$$\left. + \left(\Pr\left[v_1^{i+1} = 1 \wedge v_n^{i+1} = 1\right] + \Pr\left[v_1^i = 1 \wedge v_n^{i+1} = 0\right] - \frac{1}{2}\right)\right], \tag{6}$$

which evaluates to:

$$\begin{aligned}
&\Pr\left[v_1^0 = 1 \wedge v_n^0 = 1\right] + \Pr\left[v_1^0 = 0 \wedge v_n^r = 1\right] \\
&\quad + \ \Pr\left[v_1^0 = 0 \wedge v_n^0 = 0\right] + \Pr\left[v_1^0 = 1 \wedge v_n^r = 0\right] - 1 \\
&\quad + \ \sum_{i=0}^{r-1} \Bigg[\Pr\left[v_1^i = 1 \wedge v_n^i = 0\right] + \ \Pr\left[v_1^i = 0 \wedge v_n^i = 1\right] \\
&\qquad + \Pr\left[v_1^{i+1} = 0 \wedge v_n^{i+1} = 0\right] + \Pr\left[v_1^{i+1} = 1 \wedge v_n^{i+1} = 1\right] - 1 \Bigg] \\
&\quad = \ \Pr\left[v_1^0 = 1 \wedge v_n^0 = 1\right] + \Pr\left[v_1^0 = 0 \wedge v_n^r = 1\right] + \Pr\left[v_1^0 = 0 \wedge v_n^0 = 0\right] \\
&\qquad + \Pr\left[v_1^0 = 1 \wedge v_n^r = 0\right] + \Pr\left[v_1^0 = 1 \wedge v_n^0 = 0\right] + \Pr\left[v_1^0 = 0 \wedge v_n^0 = 1\right] \\
&\qquad + \Pr\left[v_1^r = 0 \wedge v_n^r = 0\right] + \Pr\left[v_1^r = 1 \wedge v_n^r = 1\right] - 2 \\
&\quad = \ \Pr\left[v_1^0 = 0 \wedge v_n^r = 1\right] + \Pr\left[v_1^0 = 1 \wedge v_n^r = 0\right] + \Pr\left[v_1^r = v_n^r\right] - 1 \\
&\quad \geq \ \Pr\left[v_1^0 = 0 \wedge b_1 \neq b_n\right] - \Pr\left[v_n^r \neq 1 \wedge b_1 \neq b_n\right] \\
&\qquad + \Pr\left[v_1^0 = 1 \wedge b_1 = b_n\right] - \Pr\left[v_n^r \neq 0 \wedge b_1 = b_n\right] + \Pr\left[v_1^r = v_n^r\right] - 1 \\
&\quad \geq \ \Pr\left[b_1 \neq b_n \mid v_1^0 = 0\right] \cdot \Pr[v_1^0 = 0] + \Pr\left[b_1 = b_n \mid v_1^0 = 1\right] \cdot \Pr[v_1^0 = 1] + \frac{3}{4} - 1,
\end{aligned}$$

using the assumed correctness of π when run honestly to completion. Since v_1^0 is independent of P_n's input b_n, we have $\Pr\left[b_1 \neq b_n \mid v_1^0 = 0\right] = \Pr\left[b_1 = b_n \mid v_1^0 = 1\right] = \frac{1}{2}$. It follows that the above sum is at least $\frac{1}{4}$, and so at least one of the summands (1)–(6) is at least $p(k) \stackrel{\text{def}}{=} \frac{1}{4 \cdot (4r(k)+2)}$, which is noticeable. We show that this implies that π does not securely compute f with abort in the presence of a fail-stop adversary corrupting s parties. In all the cases described below, Z continues to choose b_1, b_n uniformly and independently; however, the set of corrupted parties I and/or the auxiliary input z given to the adversary may change.

Case 1(a). Say $\Pr\left[v_1^0 = 1 \wedge v_n^0 = 1\right] + \Pr\left[v_1^0 = 0 \wedge v_n^r = 1\right] - \frac{1}{2} \geq p(k)$, and consider the adversary who corrupts parties in $A \cup B$ and does the following: it first computes v_1^0 (using the input b_1 given to it by Z and random tapes for all parties in $A \cup B$). If $v_1^0 = 1$, the adversary aborts all parties in A immediately and has parties in B run π honestly with (the honest parties) C. If $v_1^0 = 0$, the adversary has all parties in $A \cup B$ run the entire protocol honestly.

Note that $|A \cup B| = s$. Furthermore, the probability that P_n outputs 1 in a real execution of the protocol with this adversary is exactly

$$\Pr\left[v_1^0 = 1 \wedge v_n^0 = 1\right] + \Pr\left[v_1^0 = 0 \wedge v_n^r = 1\right] \geq \frac{1}{2} + p(k).$$

However, in an ideal execution with any adversary corrupting parties in $A \cup B$, the honest party P_n will not output 1 with probability greater than $\frac{1}{2}$ (given that P_n's input is chosen uniformly at random). It follows in this case that π does not securely compute f with abort in the presence of a fail-stop adversary corrupting s parties.

Case 1(b). Say $\Pr\left[v_1^0 = 0 \wedge v_n^0 = 0\right] + \Pr\left[v_1^0 = 1 \wedge v_n^r = 0\right] - \frac{1}{2} \geq p(k)$. An argument analogous to the above gives a real-world adversary who corrupts parties in $A \cup B$ and forces P_n to output 0 with probability noticeably greater than 1/2. This again implies that π does not securely compute f with abort in the presence of a fail-stop adversary corrupting s parties.

Case 2(a). Say there exists an index $i \in \{0, \ldots, r(k) - 1\}$ for which

$$\Pr\left[v_1^i = 0 \wedge v_n^{i+1} = 0\right] + \Pr\left[v_1^i = 1 \wedge v_n^i = 0\right] - \frac{1}{2} \geq p(k).$$

Consider the adversary given auxiliary input $z = i$ who corrupts the parties in $A \cup B$ and acts as follows: it runs the protocol honestly up to the end of segment i (if $i = 0$, this is just the beginning of the protocol). At this point, as noted earlier, the parties in $A \cup B$ jointly have enough information to compute v_1^i. If $v_1^i = 1$, then the adversary immediately aborts all parties in A. If $v_1^i = 0$, then the parties in A send their (honestly computed) messages for segment $i + 1$ but send no more messages after that (i.e., they abort in segment $i+2$). In either case, parties in B continue to run the entire rest of the protocol honestly.

The probability that P_n outputs 0 in a real execution of the protocol is exactly

$$\Pr\left[v_1^i = 0 \wedge v_n^{i+1} = 0\right] + \Pr\left[v_1^i = 1 \wedge v_n^i = 0\right] \geq \frac{1}{2} + p(k).$$

However, as before, in an ideal execution with any adversary corrupting parties in $A \cup B$, the honest party P_n will not output 0 with probability greater than $\frac{1}{2}$. Thus, in this case π does not securely compute f with abort in the presence of a fail-stop adversary corrupting s parties.

Case 2(b). If there exists an i such that $\Pr\left[v_1^i = 1 \wedge v_n^{i+1} = 1\right] + \Pr\left[v_1^i = 0 \wedge v_n^i = 1\right] - \frac{1}{2} \geq p(k)$, an argument as above gives an adversary corrupting parties in $A \cup B$ who forces P_n to output 1 more often than can be achieved by any adversary in the ideal world.

Case 3(a). Say there exists an index $i \in \{1, \ldots, r(k)\}$ such that

$$\Pr\left[v_1^i = 0 \wedge v_n^i = 0\right] + \Pr\left[v_1^{i-1} = 0 \wedge v_n^i = 1\right] - \frac{1}{2} \geq p(k)$$

(note that all indices have been shifted by 1 for convenience). Consider the adversary given auxiliary input $z = i$ who corrupts parties in $B \cup C$ and acts as follows: it runs the protocol honestly up to the point when it is P_{n-t+1}'s turn to send a message in segment i. (Recall that P_{n-t+1} is the party with lowest index who is in C.) At this point, the parties in $B \cup C$ can jointly compute v_n^i. If $v_n^i = 1$, then all parties in C abort in this segment and do not send any more messages (so the last messages sent by any parties in C were sent in segment $i-1$). If $v_n^i = 0$, then all parties in C send their (honestly generated) messages in segment i but abort in segment $i+1$. In either case, parties in B continue to run the entire rest of the protocol honestly.

The probability that P_1 outputs 0 in a real execution of the protocol is exactly

$$\Pr\left[v_1^i = 0 \wedge v_n^i = 0\right] + \Pr\left[v_1^{i-1} = 0 \wedge v_n^i = 1\right] \geq \frac{1}{2} + p(k).$$

However, in an ideal execution with any adversary corrupting parties in $B \cup C$, the honest party P_1 will not output 0 with probability greater than $\frac{1}{2}$ (given that its input is chosen uniformly at random). We conclude that in this case π does not securely compute f with abort in the presence of a fail-stop adversary corrupting s parties.

Case 3(b). If there exists an i such that $\Pr\left[v_1^i = 1 \wedge v_n^i = 1\right] + \Pr\left[v_1^{i-1} = 1 \wedge v_n^i = 0\right] - \frac{1}{2} \geq p(k)$, an argument as above gives an adversary corrupting parties in $B \cup C$ who forces P_1 to output 1 more often than can be achieved by any adversary in the ideal world.

3.2.2 Ruling out Privacy

The argument in the previous section shows that we cannot hope to achieve the "best of both worlds". However, we might hope that for every functionality there is a protocol π that is secure with an honest majority and is also *private* even without an honest majority. Building on the result of the previous section, we rule out this possibility as well.

Given n, t, s as before, we define a function $\tilde{f}$ that takes inputs from P_1 and P_n, and returns output to P_1, P_n, and also P_{t+1}. On input $(b_1, \alpha_0, \alpha_1)$ from P_1 and (b_n, β_0, β_1) from P_n, where $b_1, b_n, \alpha_0, \alpha_1, \beta_0, \beta_1 \in \{0,1\}$, functionality $\tilde{f}$ computes $v = b_1 \oplus b_n$, gives v to P_1 and P_n, and gives (v, α_v, β_v) to P_{t+1}. That is:

$$\begin{aligned}&\tilde{f}((b_1, \alpha_0, \alpha_1), \lambda, \ldots, \lambda, (b_n, \beta_0, \beta_1)) \stackrel{\text{def}}{=} \\ &\qquad (b_1 \oplus b_n, \lambda, \ldots, \lambda, \underbrace{(b_1 \oplus b_n, \alpha_{b_1 \oplus b_n}, \beta_{b_1 \oplus b_n})}_{\text{output of } P_{t+1}}, \lambda, \ldots, \lambda, b_1 \oplus b_n),\end{aligned}$$

where we let λ denote an empty input/output.

Let π be a protocol that securely computes $\tilde{f}$ in the presence of a malicious adversary corrupting t parties. Let A, B, C be a partition of the parties as in the previous section, and recall that $P_{t+1} \in B$. Consider an experiment in which Z chooses inputs for P_1 and P_n uniformly and independently, and all parties run protocol π honestly except that parties in A or C (but never B) may possibly abort. An argument exactly as in the previous section shows that there exists a real-world adversary $\mathcal{A}$ who either:

- corrupts the parties in $A \cup B$ and causes P_n to output some bit v with probability noticeably greater than 1/2; or
- corrupts the parties in $B \cup C$ and causes P_1 to output some bit v with probability noticeably greater than 1/2.

Assume without loss of generality that the first case holds, and there is a fail-stop adversary $\mathcal{A}$ who corrupts the s parties in $A \cup B$ and causes P_n to output 0 with probability at least $1/2 + p(k)$ for some noticeable function p. The key observation is that $\mathcal{A}$ only causes the t parties in A to abort, and the remaining corrupted parties in B continue executing the entire protocol honestly. Since π is secure in the presence of a malicious adversary corrupting t parties, all parties in $B \cup C$ will receive their outputs (except possibly with negligible probability) even if all parties in A abort. Moreover, security of π implies that the output of the honest-looking P_{t+1} will be consistent with the input and output of the honest P_n (except with negligible probability). Taken together, this means that the view of $\mathcal{A}$ — which includes the output generated by P_{t+1} — includes β_0 with probability at least $1/2 + p'(k)$ for some noticeable function p', and furthermore $\mathcal{A}$ knows when this occurs (since the output of P_{t+1} includes $v \stackrel{\text{def}}{=} b_1 \oplus b_n$ in addition to β_v). Thus, $\mathcal{A}$ can output a guess for β_0 which is correct with probability at least

$$\frac{1}{2} + p'(k) + \frac{1}{2} \cdot \left(\frac{1}{2} - p'(k)\right) = \frac{3}{4} + \frac{p'(k)}{2}\,.$$

In contrast, no ideal-world adversary $\mathcal{A}'$ corrupting $A \cup B$ can output a guess for β_0 which is correct with probability better than $3/4$ when Z chooses P_n's inputs uniformly at random. This shows that π does not privately compute $\tilde{f}$ in the presence of a malicious adversary corrupting s parties.

Remark 3.6 (Protocols with expected polynomial round complexity.) *The arguments in the previous sections can be extended to apply to protocols having* expected *polynomial round complexity. We sketch the main idea here, with respect to the proof given in Section 3.2.1. Say we have a protocol π that securely computes f and for which the expected number of segments in π is $p(k)$. Correctness of π implies that if π is run honestly to completion, then the outputs of P_1 and P_n are correct (and, in particular, agree) with all but negligible probability. Setting $r(k) = 8 \cdot r'(k)$, this means that we have $\Pr[v_1^r = v_n^r = b_1 \oplus b_n] \approx 7/8$. The remainder of the proof is as before, except that adversaries always abort by segment $r+1$ at the latest.*

4 Positive Results

In this section we explore two positive results regarding when a "best of both worlds"-type guarantee is possible. First, we briefly describe a "folklore" protocol for any (reactive or non-reactive) functionality f that is simultaneously secure against malicious adversaries corrupting any $t < n/2$ parties and secure-with-abort against malicious adversaries corrupting $n - t - 1$ parties. In light of Cleve's result [9] and Theorems 3.1 and 3.5, these thresholds are the best possible for general functionalities.

We next show a protocol whose security is incomparable to the above. Our second protocol is simultaneously secure against malicious adversaries corrupting $t < n/2$ parties as well as *semi-honest* adversaries corrupting any $s < n$ parties. (We stress that, in contrast, typical protocols offering full security against t malicious parties are completely insecure against even $t+1$ semi-honest parties.) This result applies only to non-reactive functionalities; as shown by Theorem 3.1, this is inherent.

4.1 Achieving the "Best of Both Worlds" for Suitable Thresholds

Theorem 4.1 *Let n, s, t be such that $t + s < n$ and $t < n/2$, and assume the existence of enhanced trapdoor permutations. Then for any probabilistic polynomial-time (reactive or non-reactive) functionality f, there exists a protocol that simultaneously:*

- *securely computes f in the presence of a malicious adversary corrupting t parties;*
- *securely computes f with abort in the presence of a malicious adversary corrupting s parties.*

Proof: If $s \leq t$ the theorem follows from known results [12], so assume $s > t$. We begin with the case of non-reactive functionalities. In this case, a protocol π with the claimed properties can be derived easily by suitably modifying known protocols that achieve security for honest majority. In particular, such a protocol π can be obtained by following the general approach of [12, Construction 7.5.39] with the following changes:

- The verifiable secret sharing (VSS) scheme used should have threshold $s + 1$, so that any $s + 1$ shares suffice to recover the secret but any s shares give no information (in at least a computational sense) about the secret.
- The "handling abort" procedure is modified as follows. If a party P_i aborts (or is detected cheating) at some point during the protocol, all remaining parties broadcast their shares of P_i's input and random tape. If at least $s + 1$ valid shares are revealed, the protocol continues with parties carrying out the execution on P_i's behalf. If fewer than $s + 1$ valid shares are revealed, then all parties abort the protocol with output $\perp$.

Note that when t parties are corrupted, at least $n - t \geq s + 1$ valid shares are always revealed during the "handling abort" procedure. Thus, security of π in the presence of a malicious adversary corrupting t parties follows using the same analysis as in [12]. When s parties are corrupted, they may cause the protocol to abort but cannot otherwise affect the computation since the sharing threshold is set to $s + 1$. A standard argument can be used to show that π securely computes f with abort in this case.

For reactive functionalities, we proceed as sketched in [12, Section 7.7.1.3] with natural modifications. As there, the system state s^j at the end of the jth phase of the protocol is shared among the parties; here, however, this sharing is done using threshold $s + 1$ as above. ■

4.2 Security Against a Malicious Minority or a Semi-Honest Majority

Theorem 4.2 *Let n, s, t be such that $t < n/2$ and $s < n$, and assume the existence of enhanced trapdoor permutations. Then for any probabilistic polynomial-time non-reactive functionality f, there exists a protocol that simultaneously:*

- *securely computes f in the presence of a malicious adversary corrupting t parties;*
- *securely computes f in the presence of a semi-honest adversary corrupting s parties.*

Proof: We assume $t = \lfloor (n-1)/2 \rfloor$ and $s = n-1$. We also assume without loss of generality that f is a *single-output* function, i.e., a function where all parties receive the same output. This is without loss of generality since secure computation of a general functionality $(y_1, \ldots, y_n) = \hat{f}(x_1, \ldots, x_n)$

can be reduced to secure computation of the single-output functionality $(y_1 \oplus r_1)|\cdots|(y_n \oplus r_n) \leftarrow f((r_1, x_1), \ldots, (r_n, x_n))$, where r_i is a random pad chosen by P_i at the outset of the protocol. This reduction is secure for any number of corruptions.

Before describing our protocol π computing f, we first define a related functionality $\mathcal{SS}_f$ that corresponds to computing an *authenticated secret sharing* of f (with threshold $t+1$). That is, $\mathcal{SS}_f$ denotes the functionality that

1. evaluates $y = f(x_1, \ldots, x_n)$;
2. computes a $(t+1)$-out-of-n Shamir secret sharing [21] of y, sending to each party its share y_i;
3. authenticates each share y_i for every other party P_j using an information-theoretic MAC (denoted Mac). The resulting tag is given to P_i and the key is given to P_j.

See Figure 2 for details.

Functionality $\mathcal{SS}_f$

Parameters: $\mathbb{F}$ is a finite field that includes both $[n]$ and the range of f.

Inputs: Each party P_i provides input x_i.

Computation:

1. Compute $y = f(x_1, \ldots, x_n)$ (using random coins in case f is randomized).
2. Choose random $p_1, \ldots, p_t \in \mathbb{F}$ and set $p(x) = y + \sum_{i=1}^{t} p_i x^i$.
3. Set $y_i = p(i)$ for $i \in [n]$.
4. Choose random $k_{i,j} \in \mathbb{F}$ for all $i, j, \in [n]$.
5. Compute $\mathsf{tag}_{i,j} = \mathsf{Mac}_{k_{i,j}}(y_i)$ for all $i, j \in [n]$.
6. Each party P_i is given as output y_i, $\{k_{j,i}\}_{j\in[n]}$, and $\{\mathsf{tag}_{i,j}\}_{j\in[n]}$.

Figure 2: Functionality $\mathcal{SS}_f$ for authenticated secret sharing of the output of f.

In Figure 3, we describe a protocol π computing f. This protocol relies on two sub-protocols: a sub-protocol π_n that computes $\mathcal{SS}_f$, and a sub-protocol $\pi_{n/2}$ that computes f. We require the following security guarantees from these protocols:

- π_n securely computes $\mathcal{SS}_f$ with abort in the presence of a malicious adversary corrupting s parties, and also securely computes $\mathcal{SS}_f$ in the presence of a semi-honest adversary corrupting s parties.[5]
- $\pi_{n/2}$ securely computes f in the presence of a malicious adversary corrupting t parties.

Protocols with the above properties can be constructed assuming the existence of enhanced trapdoor permutations [12].

Intuition for the claimed security properties of π is as follows. Consider first the case of a semi-honest adversary corrupting s parties in some set I. In this case, Phase I always completes

[5]Because of the way we have defined security for semi-honest adversaries, this second requirement is not implied by the first. Nevertheless, standard protocols satisfying the first requirement also satisfy the second.

Protocol π

Inputs: Each party P_i has input $(1^k, x_i)$.

Output: Each party P_i gets output $y = f(x_1, \ldots, x_n)$.

Phase I:
Each party P_i does as follows:

1. Run a protocol π_n computing the functionality $\mathcal{SS}_f$, using input x_i. Let y_i, $\{k_{j,i}\}_{j\in[n]}$, and $\{\mathsf{tag}_{i,j}\}_{j\in[n]}$ denote the output of P_i following execution of this protocol.
2. If $y_i = \perp$, then go to Phase II. Otherwise, do:
   (a) For every other party P_j, send $(y_i, \mathsf{tag}_{i,j})$ to P_j. Receive in return $(y_j, \mathsf{tag}_{j,i})$ from each other party P_j.
   (b) Set $y'_i = y_i$. For $j \neq i$, if $\mathsf{Mac}_{k_{j,i}}(y_j) = \mathsf{tag}_{j,i}$ set $y'_j = y_j$; otherwise, set $y'_j = \perp$.
   (c) Reconstruct the output y using the shares $y'_1, \ldots, y'_n$.

Phase II:
Each party P_i runs protocol $\pi_{n/2}$ computing the functionality f, using their original input x_i. Each P_i outputs the output value it obtained in π_t.

Figure 3: Protocol π, based on protocols π_s and π_t.

successfully (and so Phase II is never executed) and all honest parties learn the correct output y. Moreover, since π_n is secure in the presence of a semi-honest adversary corrupting s parties, the adversary learns nothing from the execution of π other than the secret shares $\{y_i\}_{i\in I}$, which is equivalent to learning y. Next, consider the case of a malicious adversary corrupting t parties in some set I. Here, there are two possibilities:

- **π_n completes successfully.** Protocol π_n is secure with abort in the presence of a malicious adversary corrupting $t < s$ parties. Thus, if π_n completes successfully every honest party P_i learns a (correct) share y_i of the correct output value y (in addition to correct authentication information for this share). In step 2(a) of Phase I every honest party then obtains at least $n - t > t + 1$ correct (and valid) shares of y; furthermore, an incorrect share sent by a corrupted party is detected as invalid except with negligible probability. We conclude that honest parties output the correct value y. Finally, security of π_n implies that the adversary learns nothing from this execution that is not implied by its inputs and the output value y.

- **π_n does not complete successfully.** In this case, we may assume the adversary learns its output in π_n and then aborts this sub-protocol before the honest parties learn their outputs. By security of π_n, the only thing the adversary learns from Phase I are the shares $\{y_i\}_{i\in I}$. Since these shares were generated using a secret-sharing scheme with threshold $t + 1$, they reveal nothing about the output y.

  Execution of π then continues with execution of $\pi_{n/2}$ in Phase II. Since $\pi_{n/2}$ is secure in the presence of a malicious adversary corrupting t parties, this sub-protocol completes successfully and all honest parties learn the correct output y; moreover, the adversary learns nothing from this execution that is not implied by its inputs and the output value y.

We now formalize the above.

Claim 4.3 *If π_n securely computes $\mathcal{SS}_f$ in the presence of a semi-honest adversary corrupting s parties, then π securely computes f in the presence of a semi-honest adversary corrupting s parties.*

Proof: We analyze π in a hybrid model where the parties have access to a trusted party computing $\mathcal{SS}_f$ (this trusted party computes $\mathcal{SS}_f$ according to the first ideal model, where the adversary cannot abort the computation), and show that in this hybrid model π securely computes f in the presence of a semi-honest adversary corrupting s parties. Standard composition theorems [5, 12] imply the claim.

Let $\mathcal{A}$ be a semi-honest adversary in a hybrid-model execution of π (as described above). We describe the natural semi-honest adversary $\mathcal{A}'$, running an ideal-world evaluation of f, whose behavior provides a perfect simulation of the behavior of $\mathcal{A}$. Adversary $\mathcal{A}'$ receives the set of corrupted parties $I \subset [n]$, their inputs $\{x_i\}_{i \in I}$, and auxiliary input z. It sends the inputs of the corrupted parties to the trusted party evaluating f, and receives in return an output y. Next, $\mathcal{A}'$ simply runs steps 2–6 of functionality $\mathcal{SS}_f$ (cf. Figure 2) using the value of y it obtained. The resulting values $\{(y_i, \{k_{j,i}\}_{j \in [n]}, \{\mathsf{tag}_{i,j}\}_{j \in [n]})\}_{i \in I}$ are given to $\mathcal{A}$ as the outputs of the corrupted parties from functionality $\mathcal{SS}_f$. The values $\{(y_i, \{\mathsf{tag}_{i,j}\}_{j \in I})\}_{i \notin I}$ are then given to $\mathcal{A}$ as the messages sent by the honest parties in the final round of Phase I. Finally, $\mathcal{A}'$ outputs whatever $\mathcal{A}$ does. It is straightforward to see that the joint distribution of the honest parties' outputs and the view of $\mathcal{A}$ (run as a sub-routine of $\mathcal{A}'$) in the ideal world is identical to the joint distribution of the honest parties' outputs and the view of $\mathcal{A}$ in the hybrid world. The claim follows. ■

Claim 4.4 *If π_n securely computes $\mathcal{SS}_f$ with abort in the presence of a malicious adversary corrupting t parties, and $\pi_{n/2}$ securely computes f in the presence of a malicious adversary corrupting t parties, then π securely computes f in the presence of a malicious adversary corrupting t parties.*

Proof: Once again, we analyze π in a hybrid model. Now, the parties have access to two trusted parties:

- a trusted party computing $\mathcal{SS}_f$ according to the *second* ideal model, where the adversary may prematurely abort the computation; and
- a trusted party computing f according to the *first* ideal model, where the adversary cannot abort the computation.

We show that in this hybrid model π securely computes f in the presence of a malicious adversary corrupting t parties. Standard composition theorems [5, 12] imply the claim.

Let $\mathcal{A}$ be a malicious adversary in a hybrid-model execution of π (as described above). We describe a malicious adversary $\mathcal{A}'$, running an ideal-world evaluation of f, whose behavior provides a perfect simulation of the behavior of $\mathcal{A}$. Adversary $\mathcal{A}'$ receives the set of corrupted parties $I \subset [n]$, their inputs $\{x_i\}_{i \in I}$, and auxiliary input z. It passes these values to $\mathcal{A}$, and then receives inputs $\{x'_i\}_{i \in I}$ sent by $\mathcal{A}$ to the trusted party computing $\mathcal{SS}_f$ (recall that $\mathcal{A}$ operates in the hybrid world where there is a trusted party computing this functionality). $\mathcal{A}'$ chooses random $y_i \in \mathbb{F}$ for $i \in I$, and then runs steps 4–6 of $\mathcal{SS}_f$ (cf. Figure 2). The resulting values $\{(y_i, \{k_{j,i}\}_{j \in [n]}, \{\mathsf{tag}_{i,j}\}_{j \in [n]})\}_{i \in I}$ are given to $\mathcal{A}$ as the outputs of the corrupted parties from functionality $\mathcal{SS}_f$. We stress that $\mathcal{A}'$ has not yet sent anything to its own trusted party computing f.

There are now two sub-cases, depending on whether or not $\mathcal{A}$ aborts the computation of $\mathcal{SS}_f$:

- If $\mathcal{A}$ aborts computation of $\mathcal{SS}_f$, adversary $\mathcal{A}'$ continues with simulation of Phase II as described below.

- If $\mathcal{A}$ allows computation of $\mathcal{SS}_f$ to continue, then $\mathcal{A}'$ sends the inputs $\{x'_i\}_{i\in I}$ to the trusted party computing f and receives in return a value y. It interpolates a degree-t polynomial p satisfying $p(0) = y$ and $p(i) = y_i$ for $i \in I$, and sets $y_i = p(i)$ for $i \notin I$. Finally, it gives $\{(y_i, \{\mathsf{tag}_{i,j}\}_{j\in I})\}_{i\notin I}$ to $\mathcal{A}$ as the messages sent by the honest parties in the final round of Phase I, and outputs whatever $\mathcal{A}$ outputs.

If $\mathcal{A}$ had aborted computation of $\mathcal{A}_f$, then $\mathcal{A}'$ now continues with simulation of Phase II. $\mathcal{A}$ receives (possibly different) inputs $\{x''_i\}_{i\in I}$ from $\mathcal{A}$, sends these to its trusted party computing f, and receives in return an output value y. It gives y to $\mathcal{A}$, and outputs whatever $\mathcal{A}$ outputs.

It is not hard to verify that the joint distribution of the honest parties' outputs and the view of $\mathcal{A}$ (run as a sub-routine of $\mathcal{A}'$) in the ideal world is statistically close to the joint distribution of the honest parties' outputs and the view of $\mathcal{A}$ in the hybrid world (where the only difference arises from the possibility that $\mathcal{A}$ manages to forge a tag on an invalid share). The claim follows. ■

Since protocols π_n and $\pi_{n/2}$ with the desired security properties can be constructed from enhanced trapdoor permutations, the preceding claims complete the proof of the theorem. ■

Remark 4.5 (Realizing adaptive security and/or universal composability.) *Theorem 4.2 refers to our default model of stand-alone security against static adversaries. It is not hard to see, however, that protocol π described in the proof of that theorem can be proved secure against adaptive adversaries and/or universally composable [6] if the underlying protocols $\pi_n, \pi_{n/2}$ satisfy these stronger security properties.*

References

[1] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *4th Theory of Cryptography Conference — TCC 2007*, volume 4392 of *LNCS*, pages 137–156. Springer, 2007.

[2] D. Beaver. Multiparty protocols tolerating half faulty processors. In *Advances in Cryptology — Crypto '89*, volume 435 of *LNCS*, pages 560–572. Springer, 1990.

[3] D. Beaver and S. Goldwasser. Multiparty computation with faulty majority. In *30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 468–473. IEEE, 1989.

[4] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10. ACM Press, 1988.

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

[6] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–145. IEEE, 2001.

[7] R. Canetti. Security and composition of cryptographic protocols: A tutorial (part I). *SIGACT News*, 37(3):67–92, 2006.

[8] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 11–19. ACM Press, 1988.

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

[10] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. In *Advances in Cryptology — Eurocrypt '99*, volume 1592 of *LNCS*, pages 311–326. Springer, 1999.

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

[12] O. Goldreich. *Foundations of Cryptography, vol. 2: Basic Applications*. Cambridge University Press, Cambridge, UK, 2004.

[13] S. Goldwasser and L.A. Levin. Fair computation of general functions in presence of immoral majority. In *Advances in Cryptology — Crypto '90*, volume 537 of *LNCS*, pages 77–93. Springer, 1991.

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

[15] S.D. Gordon and J. Katz. Partial fairness in secure two-party computation, 2008. Available at `http://eprint.iacr.org/2008/206`.

[16] S.D. Gordon, C. Hazay, J. Katz, and Y. Lindell. Complete fairness in secure two-party computation. In *40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 413–422. ACM Press, 2008.

[17] S.D. Gordon and J. Katz. Complete fairness in multi-party computation without an honest majority. In *6th Theory of Cryptography Conference — TCC 2009*, volume 5444 of *LNCS*, pages 19–35. Springer, 2009.

[18] Y. Ishai, E. Kushilevitz, Y. Lindell, and E. Petrank. On combining privacy with guaranteed output delivery in secure multiparty computation. In *Advances in Cryptology — Crypto 2006*, volume 4117 of *LNCS*, pages 483–500. Springer, 2006.

[19] J. Katz. On achieving the "best of both worlds" in secure multiparty computation. In *39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 11–20. ACM Press, 2007.

[20] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 73–85. ACM Press, 1989.

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

[22] A. C.-C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 162–167. IEEE, 1986.