Can Alice and Bob Guarantee Output to Carol?

Bar Alon[*]
alonbar08@gmail.com

Eran Omri[†]
omrier@ariel.ac.il

Muthuramakrishnan Venkitasubramaniam[‡]
mv783@georgetown.edu

October 17, 2023

Abstract

In the setting of solitary output computations, only a single designated party learns the output of some function applied to the private inputs of all participating parties with the guarantee that nothing beyond the output is revealed. The setting of solitary output functionalities is a special case of secure multiparty computation, which allows a set of mutually distrusting parties to compute some function of their private inputs. The computation should guarantee some security properties, such as correctness, privacy, fairness, and output delivery. Full security captures all these properties together.

Solitary output computation is a common setting that has become increasingly important, as it is relevant to many real-world scenarios, such as federated learning and set disjointness. In the set-disjointness problem, a set of parties with private datasets wish to convey to another party whether they have a common input. In this work, we investigate the limits of achieving set-disjointness which already has numerous applications and whose feasibility (under non-trivial conditions) was left open in the work of Halevi et al. (TCC 2019).

Towards resolving this, we completely characterize the set of Boolean functions that can be computed in the three-party setting in the face of a malicious adversary that corrupts up to two of the parties. As a corollary, we characterize the family of set-disjointness functions that can be computed in this setting, providing somewhat surprising results regarding this family and resolving the open question posed by Halevi et al.

[*]Department of Computer Science, Ben Gurion University.
[†]Department of Computer Science, Ariel University. Ariel Cyber Innovation Center (ACIC).
[‡]Georgetown University.

Contents

1 Introduction

Solitary output computations [26] consider a single central entity that wishes to compute a function over data that is distributed among several other entities while providing privacy of the data. Such computations are emerging as an important category and capture many real-world scenarios. Examples include service providers that wish to perform some analysis over their client's data, federated learning, federal regulatory agencies wishing to detect fraudulent users/transactions among banks, researchers looking to collect statistics from users, or a government security agency wishing to detect widespread intrusions on different high-value state agencies. In cryptography, solitary output functionalities have been considered in privacy-preserving federated learning [11, 9, 12] on the practical side, in designing minimal communication protocols via Private Simultaneous Messages Protocols [20] and its robust variant [8, 1], and in the setting of very large-scale computations for tech giants, recently introduced in [5].

From a theoretical perspective, solitary output computation is related to the question of fairness in secure multiparty computation, i.e., where it is required that either all parties obtain the output or none of them do. This may seem counterintuitive since in solitary output computation there is no issue of fairness. However, Halevi et al. [26] showed that the impossibility of computing a function f in the solitary output setting directly implies that f also cannot be computed with fairness. Our work strengthens this connection, where both our upper and lower bounds in the solitary output settings inherit ideas from the fairness literature. We believe that understanding fairness can benefit from analyzing solitary output computation.

Our work is further motivated by more practical applications. Specifically, we are interested in the special, yet important instance of implementing the *set disjointness* functionality, where one party (that has no input) wishes to learn whether a set of parties, each holding a private dataset, have a common element. Protocols for set disjointness are useful in several contexts:

1. Intelligence agencies from multiple countries trying to communicate to another country if they are all tracking the same individual (for say, deviant activities).

2. A federal health agency that wants to learn if a common disease is emerging among multiple hospitals.

3. Drug-enforcement agencies that wish to find if multiple drug stores have a common prescription.

4. A (financial) market maker that wishes to identify if two of its clients have matching orders.

We note that in most of these scenarios, it is reasonable to assume that some of the participating entities (including the central one) collude to break the privacy of honest parties' inputs or disrupt the computation. Still, it is necessary to guarantee that the central entity receives an output.

If we assume that a majority of the parties are honest, then we know since the 80s that *any* functionality can be computed with such guarantees of security [10, 33, 22]. The setting where a majority of the parties cannot be assumed to be honest is more challenging, yet an important one. The investigation of the set of solitary output functionalities that can be securely computed was initiated in the work of Halevi et al. [26]. They further considered the set disjointness functionality and showed certain parameters for which it is possible to securely compute. However, they left open the exact parameters for which the functionality can be securely computed.

We continue this line of investigation to understand the feasibility of set disjointness, but, more generally, arbitrary Boolean functionalities. We begin with understanding the already challenging three-party solitary output setting (i.e. two input parties and one output receiving party). More precisely, we ask the following concrete question:

What functions with solitary output admit full security in the three-party setting where the output receiving party has no input?

Slightly more formally, we are interested in the setting where a pair of parties Alice and Bob that hold n bit inputs x and y, respectively, with respect to a Boolean function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ and wish to securely disclose $f(x, y)$ to a third party Carol. The main question we address in this work is whether it is possible to guarantee that Carol receives an output even in the presence of an adversary that can corrupt up to any two of the parties Alice, Bob, and Carol. For the specific case of set-disjointness, Alice and Bob have inputs $\mathcal{S}$ and $\mathcal{T}$, respectively, that are subsets of some universe, and Carol wishes to learn whether $\mathcal{S} \cap \mathcal{T}$ is empty or not. It turns out this admits essentially a "trivial" protocol if we allow the parties to input the empty set. The more interesting case, and one identified in [26] as an important one, is where the parties are restricted to sets of a certain size and left the following question open:

Under what input restrictions, can the set disjointness functionality be computed with full security?

This question is formalized via the notion of secure multiparty computation [22, 34, 10, 14]. Secure multiparty computation or MPC allows a group of distrustful parties to jointly compute a function over their private inputs where nothing beyond the output is revealed. *Guaranteed output delivery* or *full security* is the strongest form of security one can demand and requires that parties learn their output in any execution. A weaker notion referred to as *fairness* only requires that no corrupted party can learn the output while simultaneously denying honest parties from receiving the output. As mentioned before, guaranteed output delivery is achievable for all functionalities if we assume an honest majority, and security hold against unbounded adversaries (with point-to-point channels and a broadcast channel) [33] or against computationally bounded adversaries (assuming a public-key infrastructure) [22, 23].

A celebrated result due to Cleve [15] says that fairness is impossible to achieve for all functionalities without an honest majority. The seminal result of [25] showed that in the two-party setting, there are non-trivial functionalities that can be computed with full security. This led to a line of works [6, 31] culminating in the work of [7] that characterized completely which Boolean functions can be computed with full security in the two-party setting (where both parties receive the same output). In the multiparty setting, much less is known when there is a dishonest majority, with only a few examples of functionalities that can be computed securely [24, 18]. Halevi et al. [26] continued this line of research to understand if full security is achievable where only one (solitary) party receives an output. Note that in this case, fairness is not an issue. Despite this fact, it was shown in [26] that even in the setting of solitary output, not all functions can be computed with full security.

A special case of solitary output is the three-party setting with Boolean functions (considered in this work) where only two parties have inputs and the third party receives the output. As part of their work, Halevi et al. [26] investigated this useful case and were able to demonstrate several possibility results. Towards analyzing the landscape of Boolean functions for which full security is achievable, they heuristically (via experiments over random functions) measure the number of functions for which the known possibility results do not hold (i.e. the status for these functions are unknown). However, their work falls short of providing a (full) characterization.

1.1 Our Results

Our main contribution is a complete characterization of the set of Boolean solitary output three-party functionalities that can be computed in our setting with guaranteed output delivery. We then use this characterization to analyze the parameters which allow set disjointness to be securely computable (see Theorem 1.3 below). Before presenting the characterization, we first introduce a new notion. The notion strengthens the notion of *semi-balanced* functionalities taken from the two-party fairness literature [31, 7].

Definition 1.1 (Strong semi-balanced, informal)**.** *Let $f : \mathcal{X} \times \mathcal{Y} \times \{\lambda\} \to \{0,1\}$ be a solitary output Boolean three-party functionality, where $|\mathcal{X}|$ and $|\mathcal{Y}|$ are polynomial in the security parameter. We call f strong semi-balanced if there exist two vectors $\mathbf{p}$ and $\mathbf{q}$ over $\mathbb{R}$, such that*

$$\begin{cases} \mathbf{M}_f^T \cdot \mathbf{p} = \mathbf{1}, \\ \sum_{x \in \mathcal{X}} p_x < 1, \end{cases} \qquad \textit{and} \qquad \begin{cases} \mathbf{M}_f \cdot \mathbf{q} = \mathbf{1}, \\ \sum_{y \in \mathcal{Y}} q_y < 1. \end{cases}$$

where $\mathbf{M}_f$ is a matrix defined as $\mathbf{M}_f(x,y) = f(x,y)$ for all x and y.

Intuitively, the vectors $\mathbf{p}$ and $\mathbf{q}$ encode strategies for the pairs (A,C) and (B,C) that include sampling an input and applying some local operation to the output of f. These strategies allow each pair to fix the output distribution for C. Additionally, as we show below, the constraints on the sum of entries in each vector bound how much information on the honest party's input an (ideal world) adversary can receive from the output alone (given that the honest party sampled its input according to the strategy). We stress that the vectors can have negative entries and for the case where the vectors only have non-negative entries, [26] showed how to securely compute the corresponding functionality.

We are now ready to state our characterization.

Theorem 1.2 (Informal, characterization of Boolean functionalities)**.** *Let $f : \mathcal{X} \times \mathcal{Y} \times \{\lambda\} \to \{0,1\}$ be a solitary output Boolean three-party functionality, where $|\mathcal{X}|$ and $|\mathcal{Y}|$ are polynomial in the security parameter. Then, if f is strong semi-balanced, it* cannot *be computed securely. On the other hand, if f is not strong semi-balanced and a secure protocol for OT exists, then f can be securely computed.*

The formal statement appears in Section 3. In Section 3.1, we provide an additional characterization, which roughly states that f can be securely computed if and only if the all-one or the all-zero vectors can be described using a specific linear combination of either the rows or columns.

As an application of Theorem 1.2, consider the disjointness functionality disj, where the domain of both A and B is $\{\mathcal{S} \subseteq \{1,2,3\} : 1 \le |\mathcal{S}| \le 2\}$. It is given by the 6×6 matrix

$$\mathbf{M}_{\mathsf{disj}} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

where the inputs are ordered lexicographically. Observe that taking $\mathbf{p} = \mathbf{q} = (1,1,1,-1,-1,-1)^T$ satisfies the conditions for being strong semi-balanced. In other words, by Theorem 1.2, disj cannot be computed securely.

On the positive side, here is a function f that can be computed with full security given by the 5×5 matrix

$$\mathbf{M}_f = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

Since $\mathbf{M}_f$ is invertible, there is a unique solution to $\mathbf{M}_f \cdot \mathbf{q} = \mathbf{1}$, namely $\mathbf{q} = (-1, 0, 1, 1, 1)^T$. As $\sum_{y \in \mathcal{Y}} q_y = 2 > 1$, it follows that f is not strong semi-balanced, hence by Theorem 1.2, the functionality f can be computed with full security. We highlight that this function serves as an example of a function whose status was unknown, i.e. not captured by the results of [26]. Indeed, the positive results of [26] capture functionalities that can be computed fairly as a *two-party* functionality, or functionalities where one of the parties can fix the output distribution using an appropriate distribution over its inputs (a class of functions [26] referred to as *forced*). The latter clearly does not hold for the above example. Additionally, there is no affine combination[1] of the rows or columns that result in the all-zero or the all-one vector, hence it cannot be computed fairly as a two-party functionality [7].

Using Theorem 1.2 we are able to analyze the disjointness functionality $\mathsf{disj}_{k,n}$, where $1 \leq k < n/2$ and the domain of both A and B is $\{\mathcal{S} \subseteq [n] : k \leq |\mathcal{S}| \leq n - k\}$.[2] We completely characterize the values of k and n for which the functionality can be computed securely, assuming the size of the domain is polynomial in the security parameter. Interestingly, this only depends on the parity of n.

Theorem 1.3. *The solitary output functionality $\mathsf{disj}_{k,n}$ can be computed securely if and only if n is even (the positive direction holds assuming a secure protocol for OT exists).*

Extensions to the multiparty setting. We also consider some natural extensions of our results to the multiparty setting with a dishonest majority. For the impossibility, it is possible to extend the result using a player partitioning argument. In more detail, let f be an $(m+1)$-party functionality, let P_0 denote the output receiving party (holding no input), and let $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_m\}$ denote the set of remaining parties. Then, if there exists a partitioning of $\mathcal{P}$ into two sets A and B of size $|\mathsf{A}|, |\mathsf{B}| \geq m/2$, such that the resulting three-party functionality is impossible to securely compute, then f cannot be computed with full security against a dishonest majority.

As a concrete example, suppose we take an $(m+1)$-party solitary output functionality that depends on only two inputs, say of P_1 and P_n. Suppose for simplicity that m is even. Then for the partition $\mathsf{A} = \{\mathsf{P}_1, \ldots, \mathsf{P}_{\frac{m}{2}}\}$ and $\mathsf{B} = \{\mathsf{P}_{\frac{m}{2}+1}, \ldots, \mathsf{P}_n\}$, if the resulting three-party functionality cannot be securely computed, it follows that the $(m+1)$-party functionality cannot be securely computed against $m/2$ corruptions. Stated differently, adding "dummy" parties does not provide any additional power if no honest majority can be guaranteed.

We note, however, that this argument does not rule out multiparty set disjointness (where all parties in $\mathcal{P}$ hold an input). This is because this problem reduces to a "degenerate" functionality

[1]An affine combination is a linear combination where the sum of coefficients is 1.

[2]If the domain of either A or B includes $\emptyset$, or $[n]$, or it contains only sets all of which have the same size, then the functionality is known to be securely computable [26].

in the three-party setting where one of the (input) parties can always fix the output, which we know can be computed securely [26]. In slightly more detail, after considering the partitioning of the m parties holding inputs into two parties, both parties hold many sets as input. This results in a function that can be securely computed since the parties can fix the output to be 1 by choosing two disjoint sets for their inputs.

On the positive side, we show an $(m + 1)$-party protocol for disjointness that is secure against $m - 1$ corruptions. In fact, this can be generalized to include all functions where any two parties among $\mathcal{P}$ can fix the output distribution by sampling their inputs accordingly. Moreover, the protocol is secure even if the output receiving party P_0 has an input, and the output of the function is not necessarily Boolean. The protocol can be further generalized to an $(m + 1)$-party protocol that is secure against $m - t + 1$ parties, assuming any $t \leq (m + 1)/2$ parties[3] among those in $\mathcal{P}$ can fix the output distribution. Note that for $t = 1$ we get the *forced* condition, where any party among $\mathcal{P}$ can fix the output distribution. This was observed by [26] to be a sufficient condition for secure computation for solitary output functionalities. Thus, we refer to the generalized set of functionalities we consider as t*-forced*. We show the following.

Theorem 1.4. *Let f be an $(m + 1)$-party solitary output t-forced functionality. Then, assuming a secure protocol for OT exists, f can be securely computed against $(m - t + 1)$ corruptions.*

The construction is a generalization of the protocol for forced functions due to [26]: The parties compute an $(m - t + 2)$-out-of-$(m + 1)$ Shamir's secret sharing of the output using a secure-with-identifiable-abort protocol. In case a party aborts, the remaining parties restart. Otherwise, all parties send their shares to P_0 to reconstruct the output. If at least t parties aborted, then P_0 sample their inputs accordingly in order to fix the output distribution.[4]

1.2 Our Techniques

We now turn to describe our techniques. We begin with our lower bound followed by our upper bound. The formal statements and proofs of these results can be found in Sections 4 and 5, respectively. Finally, we provide some high-level overview of the proof of Theorem 1.3.

General proof strategy for an impossibility result. We first present our proof strategy for the lower bounds. First, we recall a concept used in the fairness literature called *locking strategies* [32, 19]. In the two-party setting, a locking strategy for party A is a pair $(D_{\mathsf{A}}, \varphi_{\mathsf{A}})$, where D_{A} is a distribution over the set of inputs $\mathcal{X}$, and $\varphi_{\mathsf{A}} : \mathcal{X} \times \{0, 1\} \to \{0, 1\}$ is a function that determines the output of A. The function φ_{A} depends both on the input x of party A and on the value of $f(x, y)$ (which is Boolean in our setting). To be locking, the strategy is required to fix the distribution of A's output, regardless of the input y of B, i.e., the quantity $\Pr_{x \leftarrow D_{\mathsf{A}}}[\varphi_{\mathsf{A}}(x, f(x, y)) = 1]$ is independent of y. We will refer to the output of $\varphi(x, f(x, y))$ as the locked output of A. A locking strategy $(D_{\mathsf{B}}, \varphi_{\mathsf{B}})$ for B is defined analogously.

Locking strategies were used in prior works [31, 32, 19] to identify two-party functionalities f that imply fair sampling – a generalization of coin tossing. Specifically, in fair sampling, the goal for A and B is to sample correlated values from some fixed distribution. As this task is known to be

[3]Observe that for $t > (m + 1)/2$ we get an honest majority.

[4]Note that a malicious party may send an incorrect share. To overcome this issue, we let the secure-with-identifiable-abort protocol also MAC the shares and give to P_0 the key. Thus, if a malicious party modifies its share, it will be caught except with negligible probability, and this can be viewed as an abort.

impossible [2, 15], this implies that f cannot be computed securely. Specifically, the functions that imply fair sampling are the ones where the locking strategies exist for both parties and generate correlated outputs.

Now, although fairness is not an issue in the solitary output setting, we are still able to show that under certain (additional) assumptions on the locking strategies, f cannot be computed as a three-party solitary output functionality as well. We do so by making the following observation: the pair $(D_{\mathsf{A}}, \varphi_{\mathsf{A}})$ defining the locking strategy of A, also defines some correlation between the input x and the (real) output $f(x, y)$. Similarly, $(D_{\mathsf{B}}, \varphi_{\mathsf{B}})$ defines some correlation between the input y of B and the output $f(x, y)$. Moreover, as the function is already assumed to imply fair sampling, there is a correlation between the two locked outputs of the two parties, i.e., between $\varphi_{\mathsf{A}}(x, f(x, y))$ and $\varphi_{\mathsf{B}}(y, f(x, y))$, where $x \leftarrow D_{\mathsf{A}}$ and $y \leftarrow D_{\mathsf{B}}$.

Intuitively, these correlations suggest that a real-world attacker can impose some "bias" on the locked output (with respect to the honest party's strategy). Note that this "biased" output is *not* given to the parties holding the inputs (i.e., A and B). However, if the adversary also corrupts C, then this can improve its chances of guessing whatever the "biased" output would have been. This in turn leaks to the adversary some information about the honest party's input. All that is left to obtain an impossibility is to identify the functionalities for which no ideal world simulator can obtain the same information. To make the above argument formal, we first use the fact that for Boolean functions, the locking strategy for each party can be encoded using some (normalized probability) vector [31]. Next, we reduce the non-simulatability condition to constraints on the locking strategy vectors for Alice and Bob (see Definition 1.1 for the constraints). We provide more details next. Before that, we remark that, quite surprisingly, we are able to show that these conditions are, in fact, necessary by designing a secure protocol whenever the condition fails to hold.

Overview of the impossibility result. Let us fix a solitary output Boolean three-party functionality f that is strong semi-balanced. Namely, there exists vectors $\mathbf{p}$ and $\mathbf{q}$, where $\sum_{x\in\mathcal{X}} p_x < 1$ and $\sum_{y\in\mathcal{Y}} q_y < 1$, satisfying $\mathbf{M}_f \cdot \mathbf{q} = \mathbf{1}$ and $\mathbf{M}_f^T \cdot \mathbf{p} = \mathbf{1}$. Towards showing that f cannot be computed securely as a solitary output three-party functionality, it is illustrative to understand why it cannot be computed fairly as a two-party functionality.

Define vectors $\mathbf{p}'$ and $\mathbf{q}'$ to be $\mathbf{p}$ and $\mathbf{q}$ normalized with respect to the ℓ_1 norm, respectively. Then, for $\delta_1^{-1} := \sum_{x\in\mathcal{X}} |p_x|$ and for $\delta_2^{-1} := \sum_{y\in\mathcal{Y}} |q_y|$, it holds that

$$\begin{cases} \mathbf{M}_f^T \cdot \mathbf{p}' = \delta_1 \cdot \mathbf{1}, \\ \sum_{x\in\mathcal{X}} p'_x < \delta_1, \\ \sum_{x\in\mathcal{X}} |p'_x| = 1, \end{cases} \quad \text{and} \quad \begin{cases} \mathbf{M}_f \cdot \mathbf{q}' = \delta_2 \cdot \mathbf{1}, \\ \sum_{y\in\mathcal{Y}} q'_y < \delta_2, \\ \sum_{y\in\mathcal{Y}} |q'_y| = 1. \end{cases}$$

Makriyannis [31] used $\mathbf{p}'$ and $\mathbf{q}'$ to show that f cannot be computed fairly. This was done by constructing a protocol for fair sampling, which is known to be impossible [2, 15]. We next explain the construction. The vector $\mathbf{p}'$ encodes the following strategy for A: Sample an input x with probability $|p'_x|$, send it to the trusted party computing f, and flip the result received if and only if $p'_x < 0$. Similarly, $\mathbf{q}'$ encodes the following strategy for B respectively: Sample an input y with probability $|q'_y|$, send it to the trusted party, and flip the result received if and only if $q'_y < 0$. With respect to these strategies, [31] proved that the output distribution of either of the parties is independent of the input sent by the other party. Specifically, regardless of the input y' that B

sent, the output $\textsc{out}_1$ of A will be 1 with probability

$$\delta_1 + \Pr\left[\mathsf{A} \text{ flips the output}\right].$$

Similarly, the output $\textsc{out}_2$ of B is 1 with probability

$$\delta_2 + \Pr\left[\mathsf{B} \text{ flips the output}\right].$$

Finally, [31] showed that the assumptions $\sum_{x\in\mathcal{X}} p'_x < \delta_1$ and $\sum_{y\in\mathcal{Y}} q'_y < \delta_2$, imply that $\textsc{out}_1$ and $\textsc{out}_2$ are correlated.[5] This results in a secure fair sampling protocol, and thus we arrive at a contradiction.

Note that, as fairness is not an issue for solitary output functionalities, such a reduction from fair sampling is not applicable in our setting. To better explain how we overcome this issue, we next provide a direct proof that f cannot be computed fairly, and then show how to "lift" the argument to the solitary output three-party case. To simplify the discussion, we will consider perfect security.

Assume for the sake of contradiction that there exists an r-round fair two-party protocol π computing f securely (with perfect security). We let A and B sample their inputs according to the distributions encoded by $\mathbf{p}'$ and $\mathbf{q}'$, as was done in the previous construction. Define $\mathsf{flip}(x) = 1$ if $p'_x < 0$ and $\mathsf{flip}(x) = 0$ otherwise. Similarly, let $\mathsf{flip}(y) = 1$ if $q'_y < 0$ and $\mathsf{flip}(y) = 0$ otherwise.[6] First, observe that if B aborts after sending i messages to A, then the output a_i of A satisfies

$$\Pr\left[a_i \oplus \mathsf{flip}(x) = 1\right] = \delta_1 + \Pr_x\left[\mathsf{flip}(x) = 1\right].$$

This is due to the fact that in the ideal world, the output of A satisfies the above relation, as stated previously. Similarly, if A aborts after sending i messages to B, then the output b_i of B satisfies

$$\Pr\left[b_i \oplus \mathsf{flip}(y) = 1\right] = \delta_2 + \Pr_y\left[\mathsf{flip}(y) = 1\right].$$

Second, at the start of the protocol, before any message was sent, the random variables a_0 and b_0 are completely independent. Since x and y were sampled independently as well, it follows that $a_0 \oplus \mathsf{flip}(x)$ and $b_0 \oplus \mathsf{flip}(y)$ are independent. Finally, at the end of the protocol, it holds that $a_r = b_r = f(x,y)$. As stated previously, $a_r \oplus \mathsf{flip}(x)$ and $b_r \oplus \mathsf{flip}(y)$ are correlated. Thus, by an averaging argument, there exists a round where there is a "jump" in the correlation between $a_i \oplus \mathsf{flip}(x)$ and $b_i \oplus \mathsf{flip}(y)$. An adversary can then use this "jump" to bias the output of the other party. More concretely, the adversary either corrupts A and forces B to output a value $b \in \{0,1\}$ satisfying

$$\left|\Pr\left[b \oplus \mathsf{flip}(y) = 1\right] - \left(\delta_2 + \Pr_y\left[\mathsf{flip}(y) = 1\right]\right)\right| \geq \Omega(1/r),$$

or it corrupts B and forces A to output a value $a \in \{0,1\}$ satisfying

$$\left|\Pr\left[a \oplus \mathsf{flip}(x) = 1\right] - \left(\delta_1 + \Pr_x\left[\mathsf{flip}(x) = 1\right]\right)\right| \geq \Omega(1/r).$$

We now show how to "lift" the above argument to the solitary output three-party case. As stated previously, we cannot construct an adversary that biases the output, since only one party receives it. Instead, we use the fact that there is a "jump" in the correlation between $a_i \oplus \mathsf{flip}(x)$ and $b_i \oplus \mathsf{flip}(y)$. This allows the adversary to improve the probability of guessing $\mathsf{flip}(\cdot)$ applied to the

[5]In fact, to show correlation it suffices to assume $\sum_{x\in\mathcal{X}} p'_x \neq \delta_1$ and $\sum_{y\in\mathcal{Y}} q'_y \neq \delta_2$.

[6]We abuse notation and define flip over both $\mathcal{X}$ and $\mathcal{Y}$.

honest party's input. Specifically, assume without loss of generality that an adversary corrupting A and C that can force the output to be a value $b \in \{0,1\}$ satisfying

$$\Pr[b \oplus \mathsf{flip}(y) = 1] \geq (\delta_2 + \Pr_y[\mathsf{flip}(y) = 1]) + \Omega(1/r).$$

This can be done using a similar adversary to the two-party case, where A and C act honestly until the "jump", instruct A to abort depending on whether $a_i \oplus \mathsf{flip}(x) = 1$ or not, and instruct C to continue honestly until the termination of the protocol where it obtains b. Then, as the adversary holds b, it can guess $\mathsf{flip}(y) = b \oplus 1$, and it will succeed with a probability that is noticeably greater than $\delta_2 + \Pr_y[\mathsf{flip}(y) = 1]$.

To conclude the argument, we show that any ideal world simulator S cannot guess $\mathsf{flip}(y)$ with a probability that is greater than $\delta_2 + \Pr_y[\mathsf{flip}(y) = 1]$. Suppose that S sent an input x, and obtained the output $z = f(x,y)$. Let $\tilde{z} = z \oplus \mathsf{flip}(y)$. Then

$$\begin{aligned}
\Pr[\mathsf{S}(z) = \mathsf{flip}(y)] &= \Pr[\mathsf{S}(z) \oplus z = \tilde{z}] \\
&= \Pr[\mathsf{S}(0) = 0, z = 0, \tilde{z} = 0] + \Pr[\mathsf{S}(1) = 1, z = 1, \tilde{z} = 0] \\
&\quad + \Pr[\mathsf{S}(0) = 1, z = 0, \tilde{z} = 1] + \Pr[\mathsf{S}(1) = 0, z = 1, \tilde{z} = 1] \\
&= \Pr[\mathsf{S}(0) = 0] \cdot \Pr[z = 0, \tilde{z} = 0] + \Pr[\mathsf{S}(1) = 1] \cdot \Pr[z = 1, \tilde{z} = 0] \\
&\quad + \Pr[\mathsf{S}(0) = 1] \cdot \Pr[z = 0, \tilde{z} = 1] + \Pr[\mathsf{S}(1) = 0] \cdot \Pr[z = 1, \tilde{z} = 1] \\
&\leq \max\{\Pr[z = 0, \tilde{z} = 0], \Pr[z = 0, \tilde{z} = 1]\} \\
&\quad + \max\{\Pr[z = 1, \tilde{z} = 0], \Pr[z = 1, \tilde{z} = 1]\}.
\end{aligned}$$

Depending on which quantities are larger, the above expression is upper-bounded by one of the following.

- $\Pr[\tilde{z} = z] = \Pr_y[\mathsf{flip}(y) = 0]$,
- $\Pr[\tilde{z} \neq z] = \Pr_y[\mathsf{flip}(y) = 1]$,
- $\Pr[\tilde{z} = 0] = -\delta_2 + \Pr_y[\mathsf{flip}(y) = 0]$,
- $\Pr[\tilde{z} = 1] = \delta_2 + \Pr_y[\mathsf{flip}(y) = 1]$.

Recall that $\delta_2 > 0$ and $\delta_2 > \sum_{y: q'_y < 0} q'_y = \Pr_y[\mathsf{flip}(y) = 0] - \Pr_y[\mathsf{flip}(y) = 1]$. Thus, the largest quantity is $\delta_2 + \Pr_y[\mathsf{flip}(y) = 1]$, concluding the proof.

Overview of the positive results. For the positive direction, we assume that f is not strong semi-balanced. That is, either there is no vector $\mathbf{p}$ such that $\mathbf{M}_f^T \cdot \mathbf{p} = \mathbf{1}$ and $\sum_{x \in \mathcal{X}} p_x < 1$, or there is no vector $\mathbf{q}$ such that $\mathbf{M}_f \cdot \mathbf{q} = \mathbf{1}$ and $\sum_{y \in \mathcal{Y}} q_y < 1$. We may further assume without loss of generality that f *cannot* be computed securely as a *two-party* functionality. This is due to the fact that the result of [26] implies that any f that can be securely computed as a two-party functionality, can also be computed as a solitary output three-party functionality.

We use the characterization of securely computable Boolean two-party functionalities provided by [7]. They showed that such a functionality *cannot* be securely computable if and only if there is a vector $\mathbf{p}'$ such that $\mathbf{M}_f^T \cdot \mathbf{p}' = \mathbf{1}$ and $\sum_{x \in \mathcal{X}} p'_x \neq 1$, and there is a vector $\mathbf{q}'$ such that $\mathbf{M}_f \cdot \mathbf{q}' = \mathbf{1}$ and $\sum_{y \in \mathcal{Y}} q'_y \neq 1$.[7] By our assumption that f is not strong semi-balanced, the sum of entries in

[7]Such functionalities were named semi-balanced functionalities [31, 7].

either $\mathbf{p}'$ or $\mathbf{q}'$ cannot be smaller than 1. We assume the former without loss of generality and present a secure protocol for computing f. For the construction we next present, it suffices to assume there exists a vector $\mathbf{p}$ such that

$$\mathbf{M}_f^T \cdot \mathbf{p} = \mathbf{1} \quad \text{and} \quad \sum_{x \in \mathcal{X}} p_x \geq 1.$$

We next present a protocol inspired by the protocols of [25, 7], which follow the special round paradigm. Roughly, a special round i^* (whose value is unknown to all parties) is sampled at random according to a geometric distribution with a sufficiently small parameter $\alpha > 0$. Before round i^* is reached, each pair among (A, C) and (B, C) together learn a random independent value, while after i^* is reached they learn the output. Specifically, the values are held shared in a 2-out-of-2 secret sharing scheme. If A aborts at some round, then B helps C to recover the last value they received and have C output it. Similarly, if B aborts at some round, then A helps C to recover the last value they received. For this reason, these values are called *backup values*. Finally, we let the pair (A, C) learn the new (possibly random) value before the pair (B, C).

It is left to describe the distribution used to sample the random values before round i^*. For $i < i^*$ we give A and C the backup value $a_i = f(x, \tilde{y}_i)$, where $\tilde{y}_i \leftarrow \mathcal{Y}$. For B and C, if $i < i^* - 1$ then they receive $b_i = f(\tilde{x}_i, y)$, where $\tilde{x}_i \leftarrow \mathcal{X}$. If $i = i^* - 1$, then B and C receive a bit b_{i^*-1} sampled according to distribution independent of the parties' inputs.[8]

Intuitively, if the adversary corrupts C, then the only issue that could possibly arise is that of receiving two applications of f over the honest parties' input. Note, however, that in the above protocol, this will never occur, thus the protocol is private. The hardest case to analyze, is when A is corrupted. Although the adversary's view consists of only random shares, it can force C to output the bit b_{i^*-1}, which is independent of y, with noticeable probability. Nevertheless, we are able to show that if α is sufficiently small, and if we take b_{i^*-1} to be 1 with probability $\left(\sum_{x \in \mathcal{X}} p_x\right)^{-1} \leq 1$, then there exists a distribution over the inputs A that causes C to output in the ideal world, a bit that is identically distributed to the output in the real world. We refer the reader to Section 5 for more details.

Analyzing set-disjointness. As the proof is rather technical, we only provide a high-level overview of the analysis. The formal arguments can be found in Section 6. In order to apply Theorem 1.2 to the set-disjointness functionality $\mathsf{disj}_{k,n}$, we first solve the system

$$\mathbf{M}_{\mathsf{disj}} \cdot \mathbf{q} = (1, \ldots, 1)^T.$$

When the domain of both parties is $\{\mathcal{S} \subseteq [n] : k \leq |\mathcal{S}| \leq n - k\}$, we show that the (unique) solution is $\mathbf{q}_{\mathcal{T}} = (-1)^{k+|\mathcal{T}|} \cdot \binom{|\mathcal{T}|-1}{k-1}$. We do so by viewing each sum

$$\mathbf{M}_{\mathsf{disj}}(\mathcal{S}, \cdot) \cdot \mathbf{q} = \sum_{\mathcal{T} : \mathcal{S} \cap \mathcal{T} = \emptyset} q_{\mathcal{T}} = \sum_{m=k}^{n-|\mathcal{S}|} (-1)^{k+m} \cdot \binom{n - |\mathcal{S}|}{m} \binom{m-1}{k-1}$$

as a function of k, denoted $s(k)$. As it turns out, $s(k) - s(k+1) = 0$ for all $1 \leq k < n - |\mathcal{S}|$. As $s(n - |\mathcal{S}|)$ is clearly 1, the proof follows. Finally, identifying when $\mathsf{disj}_{k,n}$ can be computed reduces to analyzing $\sum_{\mathcal{S}} q_{\mathcal{S}}$ which further reduces to bounding some expressions involving binomials. See Section 6 for the details.

[8]In the protocol of [7] this bit is fixed and depends only on the function.

1.3 Related Work

The result of Cleve [15] showed that fair two-party coin tossing is impossible. Surprisingly, Gordon et al. [25] showed the possibility of obtaining fully secure protocols for non-trivial functions in the two-party setting (i.e., with no honest majority). These were substantially generalized in the works [6, 31]. The set of Boolean functionalities that are computable with full security was characterized in [7]. Finally, a systematic analysis of fair two-party computation, which includes non-Boolean and asymmetric functionalities (i.e., the parties might obtain different functions applied to their inputs), was first considered in [19, 32].

In the multiparty setting, if there is an honest majority, and the parties are given secure point-to-point channels and a broadcast channel, then full security is obtainable without any cryptographic assumptions [33]. Cohen and Lindell [16] initiated the study of the relation between fairness and guaranteed output delivery in secure multiparty computation, and showed that any functionality that can be computed with guaranteed output delivery in the broadcast model can also be computed with fairness over point-to-point channels. When the parties do not have a broadcast channel, Cohen et al. [17] characterized which *symmetric* (possibly randomized) functionalities in the point-to-point model can be computed with full security.

Alon et al. [4] extended the discussion to consider asymmetric functionalities in the point-to-point model. They provided various necessary and sufficient conditions for a functionality to be securely computable. Recently, [3] considered the solitary output case in the point-to-point model. They completely characterized the case where the output receiving party has no input, and the case where the output is one of three values (and the output receiving party might have an input).

Obtaining full security in the multiparty setting with a dishonest majority was first considered by [24], who showed that the three-party majority functionality, and n-party OR can be computed securely. Asharov et al. [7] also considered the multiparty case, where exactly half of the parties can be corrupted. The setting of a non-constant number of parties was considered in Dachman-Soled [18]. The "Best-of-both-worlds security" definition [27, 28, 29] requires full security to hold in case of an honest majority, however, if at least half of the parties are corrupted, then the same protocol should be secure-with-abort. Finally, Halevi et al. [26] were the first to consider the multiparty setting where only one party obtains the output.

Relevant to the positive results regarding fair computation in the dishonest majority setting, Lindell and Rabin [30] showed that protocols that require a round by which the parties are committed to their inputs, cannot compute any non-constant function with full security.

1.4 Organization

The preliminaries and definition of the model of computation appear in Section 2. In Section 3 we state our results. Then, in Sections 4 and 5 we prove the negative and positive results, respectively. Finally, in Section 6 we analyze the set-disjointness functionality. We discuss related works in Section 1.3.

2 Preliminaries

2.1 Notations

We use calligraphic letters to denote sets, uppercase for random variables and distributions, lowercase for values, and we use bold characters to denote vectors. For $n \in \mathbb{N}$, let $[n] = \{1, 2 \ldots n\}$. For a set $\mathcal{S}$ we write $s \leftarrow \mathcal{S}$ to indicate that s is selected uniformly at random from $\mathcal{S}$. Given a random variable (or a distribution) X, we write $x \leftarrow X$ to indicate that x is selected according to X. A PPT algorithm is probabilistic polynomial time, and a PPTM is a polynomial time (interactive) Turing machine. We let λ be the empty string, and let κ denote the security parameter. For $a, b, c \in \mathbb{R}$ we let $c \cdot [a, b]$ denote the segment $[ca, cb]$. We denote the *geometric distribution with parameter* $\alpha > 0$ as $\mathsf{Geom}(\alpha)$, and it is defined as $\Pr_{i \leftarrow \mathsf{Geom}(\alpha)}[i = n] = (1 - \alpha)^{n-1} \cdot \alpha$, for all integers $n \geq 1$.

A function $\mu \colon \mathbb{N} \to [0, 1]$ is called negligible, if for every positive polynomial $p(\cdot)$ and all sufficiently large n, it holds that $\mu(n) < 1/p(n)$. We write neg for an unspecified negligible function. For a randomized function (or an algorithm) f we write $f(x)$ to denote the random variable induced by the function on input x, and write $f(x; r)$ to denote the value when the randomness of f is fixed to r.

We let $\mathbf{1}_n$ and $\mathbf{0}_n$ denote the all-one and all-zero vectors, respectively, of dimension n. We will remove n when its value is clear from context. For a vector $\mathbf{v} \in \mathbb{R}^n$ and $i \in [n]$ we will denote its i^{th} entry by either v_i or $v(i)$. A vector is called a *probability vector* if all of its entries are positive and sum up to 1. We will sometimes treat such vectors the same as treat distributions, e.g., we write $v \leftarrow \mathbf{v}$ to indicate that v is sampled according to the distribution that corresponds to $\mathbf{v}$, namely, $v = i$ with probability v_i. For a set $\mathcal{S}$ of size n, we let $\mathbf{u}_{\mathcal{S}}$ denote the uniform probability vector of $\mathcal{S}$, defined as $u_i = 1/n$ for all $i \in \mathcal{S}$ and $u_i = 0$ otherwise. Finally, we let $|\mathbf{v}|$ denote the vector that is created by taking the absolute value in every entry of $\mathbf{v}$, i.e., its i^{th} position is $|v_i|$. For a matrix $\mathbf{M}$, we let $\mathbf{M}(\cdot, i)$ denote the i^{th} column of $\mathbf{M}$, and let $\mathbf{M}(i, \cdot)$ denote the i^{th} row. We denote its transpose by $\mathbf{M}^T$.

A *distribution ensemble* $X = \{X_{a,n}\}_{a \in \mathcal{D}_n, n \in \mathbb{N}}$ is an infinite sequence of random variables indexed by $a \in \mathcal{D}_n$ and $n \in \mathbb{N}$, where $\mathcal{D}_n$ is a domain that might depend on n. Computational indistinguishability is defined as follows.

Definition 2.1. *Let $X = \{X_{a,n}\}_{a \in \mathcal{D}_n, n \in \mathbb{N}}$ and $Y = \{Y_{a,n}\}_{a \in \mathcal{D}_n, n \in \mathbb{N}}$ be two ensembles. We say that X and Y are computationally indistinguishable, denoted $X \overset{\mathrm{C}}{\equiv} Y$, if for every non-uniform* PPT *distinguisher* D*, there exists a negligible function $\mu(\cdot)$, such that for all n and $a \in \mathcal{D}_n$, it holds that*

$$|\Pr[\mathsf{D}(X_{a,n}) = 1] - \Pr[\mathsf{D}(Y_{a,n}) = 1]| \leq \mu(n).$$

2.2 The Model of Computation

We provide the basic definitions for secure multiparty computation according to the real/ideal paradigm, for further details see [21]. Intuitively, 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.

In this paper we consider solitary output three-party functionalities. We further restrict the setting so that the output receiving party has no input. A functionality is a sequence of function

$f = \{f_\kappa\}_{\kappa\in\mathbb{N}}$, where $f_\kappa \colon \mathcal{X}_\kappa \times \mathcal{Y}_\kappa \times \{\lambda\} \to \mathcal{Z}_\kappa$ for every value of the security parameter $\kappa \in \mathbb{N}$.[9] We denote the parties by A, B and C. We let C be the output receiving party, and let A and B hold inputs x and y. To alleviate notations, we will remove κ from f and its domain and range, and simply write it as $f : \mathcal{X} \times \mathcal{Y} \times \{\lambda\} \to \mathcal{Z}$. Furthermore, all of our results hold assuming $|\mathcal{X}|$ and $|\mathcal{Y}|$ are polynomial is κ. For brevity, we will not mention it in the statements.

The Real Model

A three-party protocol π is defined by a set of three PPT interactive Turing machines A, B, and C. Each Turing machine (party) holds at the beginning of the execution the common security parameter 1^κ, a private input, and random coins. The *adversary* $\mathcal{A}$ is another PPT interactive Turing machine describing the behavior of the corrupted parties. It starts the execution with input that contains the identities of the corrupted parties, their inputs, and an additional auxiliary input aux.

The parties execute the protocol over a synchronous network. That is, the execution proceeds in rounds: each round consists of a *send phase* (where parties send their messages for this round) followed by a *receive phase* (where they receive messages from other parties). We consider a fully connected point-to-point network, where every pair of parties is connected by a communication line. We will further assume the parties have access to a broadcast channel.

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. Additionally, the adversary has full access to the view of the corrupted parties, which consists of their inputs, their random coins, and the messages they see throughout this execution. 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 its view.

Finally, we let $\text{REAL}_{\pi,\mathcal{A}(\mathsf{aux})}(\kappa,(x,y))$ denote the view of the adversary and the output of the honest parties, in an execution of π on inputs (x, y) and security parameter κ interacting with $\mathcal{A}$ with auxiliary input aux.

The Ideal Model

We consider an ideal computation with *guaranteed output delivery* (also referred to as *full security*), where a trusted party performs the computation on behalf of the parties, and the ideal-model adversary *cannot* abort the computation. An ideal computation of a solitary output three-party functionality f, on inputs $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ and security parameter κ, with an ideal-world adversary $\mathcal{A}$ running with an auxiliary input aux and corrupting a subset $\mathcal{I} \subseteq \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$ of the parties, proceeds as follows:

Parties send inputs to the trusted party: Each honest party sends its input to the trusted party. For each corrupted party, the adversary $\mathcal{A}$ sends a value v from its domain as its input. Let (x', y') denote the inputs received by the trusted party.

[9] The typical convention in secure computation is to let $f : (\{0,1\}^*)^3 \to \{0,1\}^*$. However, we will mostly be dealing with functionalities whose domain is of polynomial size in κ, which is why we introduce this notation.

The trusted party performs computation: The trusted party selects a random string r, computes $w = f(x', y'; r)$, and sends it to C. If $\mathsf{C} \in \mathcal{I}$ then $\mathcal{A}$ received w as well.

Outputs: If C is honest, then it outputs w. Otherwise, it outputs nothing. Both A and B output nothing, and the adversary $\mathcal{A}$ outputs some function of its view (i.e., the auxiliary input, its randomness, and the input and output of the corrupted parties).

We let $\text{IDEAL}_{f,\mathcal{A}(\mathsf{aux})}(\kappa, (x, y))$ denote the joint view of $\mathcal{A}$ being its output in a random execution of the above ideal-world process, and the output of the honest parties.

The Security Definition

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

Definition 2.2 (Malicious security)**.** *Let f be a three-party functionality and let π be a three-party protocol. We say that π computes f with* full security*, if for every non-uniform* PPT *adversary $\mathcal{A}$, controlling a subset $\mathcal{I} \subseteq \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$ of size at most 2 in the real world, there exists a non-uniform* PPT *adversary* Sim*, controlling the same subset $\mathcal{I}$ in the ideal-world such that*

$$\left\{\text{IDEAL}_{f,\mathsf{Sim}(\mathsf{aux})}(\kappa, (x, y))\right\}_{\kappa \in \mathbb{N}, x \in \mathcal{X}, y \in \mathcal{Y}, \mathsf{aux} \in \{0,1\}^*} \overset{\text{C}}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(\mathsf{aux})}(\kappa, (x, y))\right\}_{\kappa \in \mathbb{N}, x \in \mathcal{X}, y \in \mathcal{Y}, \mathsf{aux} \in \{0,1\}^*}.$$

We next define the notion of backup values, which are the values that C outputs in case a party aborts (after sending messages honestly). Note that this is well-defined for any fully secure protocol.

Definition 2.3 (Backup values)**.** *Let f a solitary output three-party functionality, and let π be an r-round protocol computing f with full security. Let $i \in \{0, \ldots, r\}$, sample the randomness of the parties, and consider an honest execution of π with the sampled randomness until all parties sent i messages. The i^{th} backup value of (A, C), denoted a_i, is the output of an honest C in case party B aborted after sending i messages honestly (and party A remains honest). Similarly, the i^{th} backup value of (B, C), denoted b_i, is the output of an honest C in case party A aborted after sending i messages honestly.*

2.3 The Dealer Model

In the description of our positive results, it will be convenient to consider the dealer model. Here, the real world is augmented with a trusted dealer that can interact with the parties in a limited way. Additionally, the adversary is assumed to be fail-stop, namely, it acts honestly, however, it may decide to abort prematurely. Essentially, the dealer will compute the backup values for (A, C) and for (B, C). In each round, the dealer sends to each pair its corresponding backup value, held shared by the two parties using a 2-out-of-2 secret sharing. After receiving its shares, an adversary can decide whether to abort a (corrupted) party or not. If a party aborts, then the dealer notifies the other parties and halts.

We note that removing the dealer can be done using standard techniques. Specifically, at the beginning of the interaction, the parties will compute a secret sharing of all backup values, where the schemes are all 3-out-of-3. Then, in each round, B broadcasts its share of (A, C)'s backup value, followed by A broadcasting its share of (B, C)'s backup value. To ensure the parties send the correct shares, the shares are also signed using a (one-time) MAC, which is then verified by the other parties. Thus, sending an incorrect share is caught except with negligible probability, and can be viewed as an abort.

2.4 Definitions From The Fairness Literature

We will use certain notions that were defined in order to analyze fairness in the two-party setting. We first associate a matrix with any Boolean solitary output three-party functionality, where one of the parties has no input. Throughout the rest of the section, we fix $f : \mathcal{X} \times \mathcal{Y} \times \{\lambda\} \to \{0, 1\}$ to be a Boolean solitary output three-party function.

Definition 2.4 (The matrix associated with a function)**.** *We associate with f an $|\mathcal{X}| \times |\mathcal{Y}|$ Boolean matrix $\mathbf{M}_f$, defined as $\mathbf{M}_f(x, y) = f(x, y)$ for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.*

A function is called forced if either A or B can fix the output distribution using an appropriate distribution over their inputs.

Definition 2.5 (Forced function [26])**.** *We say that f is forced if either there exists a probability vector $\mathbf{p}$ such that $\mathbf{p}^T \cdot \mathbf{M}_f = \delta_1 \cdot \mathbf{1}^T$, for some $\delta_1 \geq 0$, or there exists a probability vector $\mathbf{q}$ such that $\mathbf{M}_f \cdot \mathbf{q} = \delta_2 \cdot \mathbf{1}$, for some $\delta_2 \geq 0$.*

The next definition describes locking strategies for Boolean functionalities. Roughly speaking, a locking strategy [19, 32] for a party is a way for it to sample an input, and apply a local operation to the output of the function, such that the distribution of its final output is independent of the other party's input.

For Boolean functions, the local operation is to either flip or not, possibly depending on the input. Makriyannis [31] encoded these strategies (for each party) using a single vector, where the absolute value of each entry represents the weight of the corresponding input, and the sign represents whether or not the party should flip the output.

Definition 2.6 (Locking strategy for Boolean functionalities [31, 19, 32])**.** *A* locking strategy for A *is a vector $\mathbf{p} \in \mathbb{R}^{|\mathcal{X}|}$ satisfying $\mathbf{p}^T \cdot \mathbf{M}_f = \delta \cdot \mathbf{1}^T$, for some $\delta \in \mathbb{R}$. We call the locking strategy $\mathbf{p}$* normalized *if $\sum_{x \in \mathcal{X}} |p_x| = 1$.*[10] *Similarly, a* locking strategy for B *is a vector $\mathbf{q} \in \mathbb{R}^{|\mathcal{Y}|}$ satisfying $\mathbf{M}_f \cdot \mathbf{q} = \delta \cdot \mathbf{1}$, for some $\delta \in \mathbb{R}$. We call $\mathbf{q}$* normalized *if $\sum_{y \in \mathcal{Y}} |q_y| = 1$. Observe that the set of all locking strategies for A (and B) forms a linear subspace.*

We next define semi-balanced functionalities. These are functionalities where, in addition to both A and B having locking strategies, the resulting two (possibly flipped) outputs are correlated. As shown by Makriyannis [31], the latter condition have can be expressed as a condition on the total weight of the vectors representing the locking strategies.

Definition 2.7 (Semi-balanced functionalities [31, 7])**.** *We call f* semi-balanced *if there exists locking strategies $\mathbf{p}$ and $\mathbf{q}$ for A and B, respectively, such that*

$$\begin{cases} \mathbf{p}^T \cdot \mathbf{M}_f = \mathbf{1}^T, \\ \mathbf{1}^T \cdot \mathbf{p} \neq 1, \end{cases} \quad \text{and} \quad \begin{cases} \mathbf{M}_f \cdot \mathbf{q} = \mathbf{1}, \\ \mathbf{1}^T \cdot \mathbf{q} \neq 1. \end{cases}$$

[10] [19, 32] defined locking strategies as the algorithm themselves, rather then their encodings.

3 Statement of Our Results

Our main result is the complete characterization of the Boolean solitary output three-party functionalities that can be computed with full security. Roughly, our characterization state that a functionality *cannot* be computed securely if and only if it is semi-balanced, where the locking strategies of both parties have weights less than 1. We call these functionalities strong semi-balanced.

Definition 3.1 (Strong semi-balanced)**.** *Let $f : \mathcal{X} \times \mathcal{Y} \times \{\lambda\} \to \{0,1\}$ be a solitary output three-party functionality. Suppose that f is semi-balanced with associated locking strategies $\mathbf{p}$ and $\mathbf{q}$. We call f* strong semi-balanced *if $\mathbf{1}^T \cdot \mathbf{p} < 1$ and $\mathbf{1}^T \cdot \mathbf{q} < 1$.*

Theorem 3.2. *Let $f : \mathcal{X} \times \mathcal{Y} \times \{\lambda\} \to \{0,1\}$ be a solitary output three-party functionality. Then, if f is strong semi-balanced, then it cannot be computed with full security. On the other hand, if a secure protocol for OT exists, and f is not strong semi-balanced, then it can be computed with full security.*

Remark 3.3. *Although only stated and proved for deterministic functionalities, Theorem 3.2 can be easily generalized to randomized functionalities by defining $\mathbf{M}_f(x,y) = \Pr[f(x,y)=1]$ for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.*

The proof of Theorem 3.2 follows from the following three lemmata. The first lemma states the impossibility of computing strong semi-balanced. The second lemma, is a combination of the result of Halevi et al. [26] stating that any two-party functionality that can be computed fairly, can also be computed as a solitary output three-party functionality, and the result of Asharov et al. [7] which identifies the non-semi-balanced as the only functionalities that can be computed fairly. Finally, the third lemma states that the remaining set of functionalities can be computed securely.

Lemma 3.4. *Let $f : \mathcal{X} \times \mathcal{Y} \times \{\lambda\} \to \{0,1\}$ be a strong semi-balanced solitary output three-party functionality. Then f cannot be computed with full security.*

Lemma 3.5 ([26, Theorem 4.1] and [7])**.** *Let $f : \mathcal{X} \times \mathcal{Y} \times \{\lambda\} \to \{0,1\}$ be a solitary output three-party non-semi-balanced functionality. Then, if a secure protocol for OT exists, f can be computed with full security.*

Lemma 3.6. *Let $f : \mathcal{X} \times \mathcal{Y} \times \{\lambda\} \to \{0,1\}$ be a solitary output three-party Boolean functionality. Assume there exists a locking strategy $\mathbf{p}$ for A satisfying*

$$\mathbf{p}^T \cdot \mathbf{M}_f = \mathbf{1}^T \quad \text{and} \quad \mathbf{1}^T \cdot \mathbf{p} \geq 1,$$

or there exists a locking strategy $\mathbf{q}$ for B satisfying

$$\mathbf{M}_f \cdot \mathbf{q} = \mathbf{1} \quad \text{and} \quad \mathbf{1}^T \cdot \mathbf{q} \geq 1.$$

Then, if a secure protocol for OT exists, f can be computed with full security.

Proof of Theorem 3.2. The negative direction is immediately implied by Lemma 3.4. For the other direction, let us assume that f is not strong semi-balanced. By Lemma 3.5, we may assume that f is semi-balanced as otherwise f can be computed with full security. Therefore, there exist locking strategies $\mathbf{p}$ and $\mathbf{q}$ for A and B, respectively, such that

$$\begin{cases} \mathbf{p}^T \cdot \mathbf{M}_f = \mathbf{1}^T, \\ \mathbf{1}^T \cdot \mathbf{p} \neq 1, \end{cases} \quad \text{and} \quad \begin{cases} \mathbf{M}_f \cdot \mathbf{q} = \mathbf{1}, \\ \mathbf{1}^T \cdot \mathbf{q} \neq 1. \end{cases}$$

As f is not strong semi-balanced, the sum of entries in one of the above vectors must be greater than 1. Thus, by Lemma 3.6, f can be computed with full security. $\square$

We prove Lemmas 3.4 and 3.6 in Sections 4 and 5, respectively. To illustrate Theorem 3.2, we use it to analyze the parameters for which the *disjointness* functionality can be computed with full security. For parameters $n, k \in \mathbb{N}$, where $1 \leq k < n/2$, we define the solitary output three-party functionality disj as

$$\mathsf{disj}\left(\mathcal{S}, \mathcal{T}\right) = \begin{cases} 1 & \text{if } \mathcal{S} \cap \mathcal{T} = \emptyset \\ 0 & \text{otherwise} \end{cases}$$

where the domain of both A and B is $\{\mathcal{S} \subseteq [n] : k \leq |\mathcal{S}| \leq n - k\}$. We use Theorem 3.2 to prove the following.

Theorem 3.7. *Assuming a secure protocol for OT exists,* disj *can be computed with full security if and only if n is even.*

The proof is given in Section 6.

3.1 An Equivalent Characterization

Similarly to the characterization of [7] for two-party fairness, we can provide the following equivalent condition. Roughly, it states that f can be computed securely if and only if $\mathbf{0}$ or $\mathbf{1}$ can be described using a specific linear combination of either the rows or columns.

Theorem 3.8 (Equivalent characterization)**.** *Let $f : \mathcal{X} \times \mathcal{Y} \times \{\lambda\} \to \{0, 1\}$ be a solitary output three-party functionality. Then, if a secure protocol for OT exists, f can be computed with full security if and only if f is not strong semi-balanced if and only if one of the following hold.*

1. *$\mathbf{0}^T$ is an affine combination of the rows of $\mathbf{M}_f$.*
2. *$\mathbf{1}$ is a linear combination of the columns of $\mathbf{M}_f$, where the sum of coefficients is at least 1.*
3. *$\mathbf{1}^T$ is a linear combination of the rows of $\mathbf{M}_f$, where the sum of coefficients is at least 1.*
4. *$\mathbf{0}$ is an affine combination of the columns of $\mathbf{M}_f$.*

Towards proving Theorem 3.8, we use the following proposition proved by [7].

Proposition 3.9 ([7, Proposition 2.1])**.** *For any matrix $\mathbf{M}$, it holds that $\mathbf{0}^T$ is an affine combination of the rows of $\mathbf{M}$ if and only if $\mathbf{1}$ is* not *a linear combination of the columns of $\mathbf{M}$.*

Proof of Theorem 3.8. We first show that if f is not strong semi-balanced then one of Items 1 to 4 hold. Assume without loss of generality there is no locking strategy $\mathbf{q}$ for B such that

$$\mathbf{M}_f \cdot \mathbf{q} = \mathbf{1} \quad \text{and} \quad \mathbf{1}^T \cdot \mathbf{q} < 1.$$

The case where there is no locking strategy $\mathbf{p}$ for A is analogous. We show that either Item 1 or Item 2 hold. Indeed, either $\mathbf{1}$ is not a linear combination of the columns of $\mathbf{M}_f$, or it is a linear combination with the sum coefficient being at least 1. By Proposition 3.9, the former case implies that $\mathbf{0}^T$ is an affine combination of the rows of $\mathbf{M}$, thus concluding this direction.

To conclude the proof, we show that if one of Items 1 to 4 hold, then f is not strong semi-balanced. We show the proof assuming either Item 1 or Item 2 hold, as the other two cases are analogous. First, if $\mathbf{0}^T$ is an affine combination of the rows of $\mathbf{M}_f$, then by Proposition 3.9, $\mathbf{1}$ is *not* a linear combination of the columns of $\mathbf{M}_f$. Thus, there is no locking strategy for B. Assume now that there exists $\mathbf{q}$ such that

$$\mathbf{M}_f \cdot \mathbf{q} = \mathbf{1} \quad \text{and} \quad \mathbf{1}^T \cdot \mathbf{q} \geq 1.$$

The fact that f is not strong semi-balanced is a direct application of Lemma 3.6 and Theorem 3.2.

We next present a different argument that does not rely on the existence of a secure protocol, which in itself assumes the existence of oblivious transfer. Assume towards contradiction that f is strong semi-balanced. Then there exists locking strategies $\mathbf{p}$ and $\mathbf{q}'$ for A and B, respectively, satisfying

$$\begin{cases} \mathbf{p}^T \cdot \mathbf{M}_f = \mathbf{1}^T, \\ \mathbf{1}^T \cdot \mathbf{p} < 1, \end{cases} \quad \text{and} \quad \begin{cases} \mathbf{M}_f \cdot \mathbf{q}' = \mathbf{1}, \\ \mathbf{1}^T \cdot \mathbf{q}' < 1. \end{cases}$$

By considering $\mathbf{p}^T \cdot \mathbf{M}_f \cdot \mathbf{q}'$, it follows that $\mathbf{p}^T \cdot \mathbf{1} = \mathbf{1}^T \cdot \mathbf{q}'$. Similarly, by considering $\mathbf{p}^T \cdot \mathbf{M}_f \cdot \mathbf{q}$, it follows that $\mathbf{p}^T \cdot \mathbf{1} = \mathbf{1}^T \cdot \mathbf{q}$. Therefore,

$$1 \leq \mathbf{1}^T \cdot \mathbf{q} = \mathbf{1}^T \cdot \mathbf{q}' < 1,$$

which is clearly a contradiction. □

4 Impossibility of Computing Strong Semi-Balanced Functionalities

In this section, we prove our impossibility results, showing that strong semi-balanced cannot be computed with full security.

Lemma 4.1 (Restatement of Lemma 3.4)**.** *Let $f : \mathcal{X} \times \mathcal{Y} \times \{\lambda\} \to \{0,1\}$ be a strong semi-balanced solitary output three-party functionality. Then f cannot be computed with full security.*

Proof. The proof is done in two steps. In the first step, we show that for any protocol computing f, there exists an adversary that can "bias" a certain correlation between the honest party's input, and the backup value corresponding to C and the honest party. We then show how this "bias" allows the adversary to increase its chances of guessing a certain property of the honest party's input. In the second step, we show that no simulator can do better. We next formalize the above proof strategy.

Assume towards contradiction that there exists a secure r-round protocol π computing f. We show there exists an adversary that cannot be simulated. We assume that each round is composed of 3 broadcast messages, the first sent by A, the second sent by B, and the third by C (this is without loss of generality, as we allow the adversary to be rushing). Fix two normalized locking strategies $\mathbf{p} \in \mathbb{R}^{|\mathcal{X}|}$ and $\mathbf{q} \in \mathbb{R}^{|\mathcal{Y}|}$ for A and B, respectively, with respect to which f is strong semi-balanced. That is, it holds that

$$\begin{cases} \mathbf{p}^T \cdot \mathbf{M}_f = \delta_1 \cdot \mathbf{1}^T, \text{ where } \delta_1 > 0 \\ \mathbf{1}^T \cdot \mathbf{p} < \delta_1, \\ \sum_{x \in \mathcal{X}} |p_x| = 1, \end{cases} \quad \text{and} \quad \begin{cases} \mathbf{M}_f \cdot \mathbf{q} = \delta_2 \cdot \mathbf{1}, \text{ where } \delta_2 > 0 \\ \mathbf{1}^T \cdot \mathbf{q}_y < \delta_2, \\ \sum_{y \in \mathcal{Y}} |q_y| = 1. \end{cases}$$

Throughout the rest of the proof, we consider an execution of π where the inputs x and y of A and B, respectively, are sampled independently according to $|\mathbf{p}|$ and $|\mathbf{q}|$, respectively. That is, A holds input x with probability $|p_x|$ and B holds input y with probability $|q_y|$.

Let us first introduce some notations. Let $\mathsf{flip}(x)$ output 0 if $p_x \geq 0$ and output 1 otherwise, and let $\mathsf{flip}(y)$ output 0 if $q_y \geq 0$ and output 1 otherwise. Let $p^- = 1 - p^+ = \sum_{x\in\mathcal{X}:p_x<0} |p_x|$ denote the probability that $\mathsf{flip}(x) = 1$ and let $q^- = 1 - q^+ = \sum_{y\in\mathcal{Y}:q_y<0} |q_y|$ denote the probability that $\mathsf{flip}(y) = 1$. Next, recall that in Definition 2.3, for every $i \in \{0, \ldots, r\}$ we let a_i denote the backup value of and A and C, being the output of C in case B aborts after A sent i messages. Similarly, we let b_i denote the backup value of and B and C, being the output of C in case A aborts after B sent i messages.

Recall that the idea is to let the adversary "bias" a certain correlation between the backup value and the honest party's input. We define this correlation to be the backup values, that are possibly flipped, depending on the value of $\mathsf{flip}(\cdot)$. Formally, for every $i \in \{0, \ldots, r\}$ we let $\tilde{a}_i = a_i \oplus \mathsf{flip}(x)$ and we let $\tilde{b}_i = b_i \oplus \mathsf{flip}(y)$.

The next two lemmata formalize the aforementioned two steps of the proof strategy. Formally, the first lemma states that there exists a real world adversary that can "bias" either $\tilde{a}_i$ or $\tilde{b}_i$, thus allowing it to slightly improve its probability of guessing $\mathsf{flip}(\cdot)$ applied to the honest party's input. The second lemma states that no ideal world simulator can guess this value as well as the real world adversary.

Lemma 4.2. *There exists a constant $\xi > 0$ (independent of the protocol) such that one of the following holds.*

- *There exists an adversary corrupting A and C that can guess $\mathsf{flip}(y)$ with probability at least $\delta_2 + q^- + \xi/r$.*
- *There exists an adversary corrupting B and C that can guess $\mathsf{flip}(x)$ with probability at least $\delta_1 + p^- + \xi/r$.*

Lemma 4.3. *For any (possibly randomized) algorithm $\mathsf{S}_\mathsf{A} : \mathcal{X} \times \{0,1\} \to \{0,1\}$ and every $x \in \mathcal{X}$, it holds that*

$$\Pr_{y\leftarrow|\mathbf{q}|}\left[\mathsf{S}_\mathsf{A}(x, f(x,y)) = \mathsf{flip}(y)\right] \leq \delta_2 + q^-.$$

Similarly, for any (possibly randomized) algorithm $\mathsf{S}_\mathsf{B} : \mathcal{Y} \times \{0,1\} \to \{0,1\}$ and every $y \in \mathcal{Y}$, it holds that

$$\Pr_{x\leftarrow|\mathbf{p}|}\left[\mathsf{S}_\mathsf{B}(y, f(x,y)) = \mathsf{flip}(x)\right] \leq \delta_1 + p^-.$$

Clearly, Lemma 4.1 is implied by the above two lemmata. It is left to prove them. For both lemmata, we will use the following properties, proved by [31], that any semi-balanced functionality satisfy.

Lemma 4.4. *Let $f : \mathcal{X} \times \mathcal{Y} \times \{\lambda\} \to \{0,1\}$ be a semi-balance solitary output three-party functionality, with normalized locking strategies $\mathbf{p}$ and $\mathbf{q}$ for A and B, respectively. Let δ_1, δ_2, $\mathsf{flip}(\cdot)$, p^-, p^+, q^-, and q^+ be as above. Then the following hold.*

1. *[31, Lemma 6.4:] $(p^+ - p^-)\delta_2 = (q^+ - q^-)\delta_1$.*
2. *[31, Lemma 6.5:] For all $y \in \mathcal{Y}$ it holds that $\Pr_{x\leftarrow|\mathbf{p}|}\left[f(x,y) \oplus \mathsf{flip}(x) = 1\right] = \delta_1 + p^-$.*
3. *[31, Lemma 6.5:] For all $x \in \mathcal{X}$ it holds that $\Pr_{y\leftarrow|\mathbf{q}|}\left[f(x,y) \oplus \mathsf{flip}(y) = 1\right] = \delta_2 + q^-$.*

We first prove Lemma 4.2.

Proof of Lemma 4.2. We first use Items 2 and 3 of Lemma 4.4 to show that the distributions of every $\tilde{a}_i$ and every $\tilde{b}_i$ are fixed throughout the execution of π.

Claim 4.5. *For every $i \in \{0, \ldots, r\}$ it holds that*

$$\left|\Pr\left[\tilde{a}_i = 1\right] - (\delta_1 + p^-)\right| = \operatorname{neg}(\kappa) \quad \textit{and} \quad \left|\Pr\left[\tilde{b}_i = 1\right] - (\delta_2 + q^-)\right| = \operatorname{neg}(\kappa).$$

Proof. Fix $i \in \{0, \ldots, r\}$. We show only the first assertion (the second is analogous). Consider an adversary $\mathcal{B}$ corrupting (only) B, that aborts after receiving i messages from A. Then the output of an honest C is a_i. Since π is assumed to be secure, there exists a simulator $\mathsf{Sim}_{\mathcal{B}}$ for $\mathcal{B}$. By Item 2 of Lemma 4.4, in the ideal world it holds that $\Pr\left[\textsc{out}^{\mathsf{ideal}}(x, y) \oplus \mathsf{flip}(y)\right] = \delta_1 + p^-$, regardless of what $\mathsf{Sim}_{\mathcal{B}}$ sends to the trusted party. Therefore, up to a negligible difference, the same holds in the real world. □

Observe that as $\delta_1 > \mathbf{1}^T \cdot \mathbf{p} = p^+ - p^-$ and $\delta_1 > 0$, it follows that $\delta_1 + p^- \geq -\delta_1 + p^+$ as well. Similarly, it holds that $\delta_2 + q^- \geq -\delta_2 + q^+$. The next claim asserts that at some round i, there is a "jump" in the distribution of the (possibly flipped) backup values.

Claim 4.6. *There exists a value $z \in \{0, 1\}$ and a constant $\xi > 0$ (independent of the protocol), such that one of the following holds. Either there exists a round $i \in [r]$ such that*

$$\left|\Pr\left[\tilde{a}_i = z \wedge \tilde{b}_{i-1} = 1\right] + \Pr\left[\tilde{a}_i \neq z \wedge \tilde{b}_i = 1\right] - (\delta_2 + q^-)\right| \geq \xi / r,$$

or

$$\left|\Pr\left[\tilde{b}_{i-1} = z \wedge \tilde{a}_{i-1} = 1\right] + \Pr\left[\tilde{b}_{i-1} \neq z \wedge \tilde{a}_i = 1\right] - (\delta_1 + p^-)\right| \geq \xi / r.$$

Proof. By Claim 4.5, for every $i \in [r]$ and every $z \in \{0, 1\}$ we have:

$$\begin{aligned} &\Pr\left[\tilde{a}_i = z \wedge \tilde{b}_{i-1} = 1\right] + \Pr\left[\tilde{a}_i \neq z \wedge \tilde{b}_i = 1\right] - (\delta_2 + q^-) \\ &\quad \geq \Pr\left[\tilde{a}_i = z \wedge \tilde{b}_{i-1} = 1\right] + \Pr\left[\tilde{a}_i \neq z \wedge \tilde{b}_i = 1\right] - \Pr\left[\tilde{b}_i = 1\right] - \operatorname{neg}(\kappa) \\ &\quad = \Pr\left[\tilde{a}_i = z \wedge \tilde{b}_{i-1} = 1\right] - \Pr\left[\tilde{a}_i = z \wedge \tilde{b}_i = 1\right] - \operatorname{neg}(\kappa) \end{aligned}$$

Similarly, it holds that

$$\begin{aligned} &\Pr\left[\tilde{b}_{i-1} = z \wedge \tilde{a}_{i-1} = 1\right] + \Pr\left[\tilde{b}_{i-1} \neq z \wedge \tilde{a}_i = 1\right] - (\delta_1 + p^-) \\ &\quad \geq \Pr\left[\tilde{b}_{i-1} = z \wedge \tilde{a}_{i-1} = 1\right] - \Pr\left[\tilde{b}_{i-1} = z \wedge \tilde{a}_i = 1\right] - \operatorname{neg}(\kappa). \end{aligned}$$

Let Δ denote the average of the absolute values of the above quantity, taken over all i's and z's, i.e.,

$$\begin{aligned} \Delta := \frac{1}{4r} \sum_{i=1}^{r} \sum_{z=0}^{1} \Bigg[&\left|\Pr\left[\tilde{a}_i = z \wedge \tilde{b}_{i-1} = 1\right] - \Pr\left[\tilde{a}_i = z \wedge \tilde{b}_i = 1\right] - \operatorname{neg}(\kappa)\right| \\ &+ \left|\Pr\left[\tilde{b}_{i-1} = z \wedge \tilde{a}_{i-1} = 1\right] - \Pr\left[\tilde{b}_{i-1} = z \wedge \tilde{a}_i = 1\right] - \operatorname{neg}(\kappa)\right| \Bigg]. \end{aligned}$$

Observe that

$$\begin{aligned}
4r \cdot \Delta = \sum_{i=1}^{r} \Bigg[&\left|\Pr\left[\tilde{a}_i = 0 \wedge \tilde{b}_{i-1} = 1\right] - \Pr\left[\tilde{a}_i = 0 \wedge \tilde{b}_i = 1\right] - \mathrm{neg}(\kappa)\right| \\
&+ \left|\Pr\left[\tilde{b}_{i-1} = 0 \wedge \tilde{a}_{i-1} = 1\right] - \Pr\left[\tilde{b}_{i-1} = 0 \wedge \tilde{a}_i = 1\right] - \mathrm{neg}(\kappa)\right| \\
&+ \left|\Pr\left[\tilde{a}_i = 1 \wedge \tilde{b}_{i-1} = 1\right] - \Pr\left[\tilde{a}_i = 1 \wedge \tilde{b}_i = 1\right] - \mathrm{neg}(\kappa)\right| \\
&+ \left|\Pr\left[\tilde{b}_{i-1} = 1 \wedge \tilde{a}_{i-1} = 1\right] - \Pr\left[\tilde{b}_{i-1} = 1 \wedge \tilde{a}_i = 1\right] - \mathrm{neg}(\kappa)\right| \Bigg] \\
= \sum_{i=1}^{r} \Bigg[&\left|\Pr\left[\tilde{a}_i = 0 \wedge \tilde{b}_{i-1} = 0\right] - \Pr\left[\tilde{a}_i = 0 \wedge \tilde{b}_i = 0\right] + \mathrm{neg}(\kappa)\right| \\
&+ \left|\Pr\left[\tilde{b}_{i-1} = 0 \wedge \tilde{a}_{i-1} = 0\right] - \Pr\left[\tilde{b}_{i-1} = 0 \wedge \tilde{a}_i = 0\right] + \mathrm{neg}(\kappa)\right| \\
&+ \left|\Pr\left[\tilde{a}_i = 1 \wedge \tilde{b}_{i-1} = 1\right] - \Pr\left[\tilde{a}_i = 1 \wedge \tilde{b}_i = 1\right] + \mathrm{neg}(\kappa)\right| \\
&+ \left|\Pr\left[\tilde{b}_{i-1} = 1 \wedge \tilde{a}_{i-1} = 1\right] - \Pr\left[\tilde{b}_{i-1} = 1 \wedge \tilde{a}_i = 1\right] + \mathrm{neg}(\kappa)\right| \Bigg] \\
\geq \sum_{i=1}^{r} \Bigg[&\left|\Pr\left[\tilde{a}_i = \tilde{b}_{i-1}\right] - \Pr\left[\tilde{a}_i = \tilde{b}_i\right] + 2\,\mathrm{neg}(\kappa)\right| \\
&+ \left|\Pr\left[\tilde{b}_{i-1} = \tilde{a}_{i-1}\right] - \Pr\left[\tilde{b}_{i-1} = \tilde{a}_i\right] + 2\,\mathrm{neg}(\kappa)\right| \Bigg] \\
\geq &\left|\Pr\left[\tilde{a}_0 = \tilde{b}_0\right] - \Pr\left[\tilde{a}_r = \tilde{b}_r\right]\right| - 2r \cdot \mathrm{neg}(\kappa).
\end{aligned}$$

where the inequalities follow from the triangle inequality.

Now, since $\tilde{a}_0$ and $\tilde{b}_0$ are computed before any interaction is done, they are independent. Thus, by Claim 4.5 it follows that

$$\left|\Pr\left[\tilde{a}_0 = \tilde{b}_0\right] - \Big((\delta_1 + p^-)(\delta_2 + q^-) + (-\delta_1 + p^+)(-\delta_2 + q^+)\Big)\right| = \mathrm{neg}(\kappa).$$

Additionally, since $\tilde{a}_r$ and $\tilde{b}_r$ correspond to the (possibly flipped) output of the protocol, it follows that they are equal if and only $\mathsf{flip}(x) = \mathsf{flip}(y)$. Thus,

$$\left|\Pr\left[\tilde{a}_r = \tilde{b}_r\right] - \Big(p^- q^- + p^+ q^+\Big)\right| = \mathrm{neg}(\kappa).$$

Therefore,

$$\Delta \geq \frac{1}{4r}\left|2\delta_1\delta_2 + (q^- - q^+)\delta_1 + (p^- - p^+)\delta_2\right| - \mathrm{neg}(\kappa).$$

By Item 1 of Lemma 4.4, it holds that $(q^- - q^+)\delta_1 = (p^- - p^+)\delta_2$, hence

$$\Delta \geq \frac{\delta_1}{2r}\left|\delta_2 + q^- - q^+\right| - \mathrm{neg}(\kappa).$$

Since $\delta_1 \neq 0$ and $\delta_2 \neq \mathbf{1}^T \cdot \mathbf{q} = q^+ - q^-$, it follows that for $\xi := \delta_1 \cdot \left|\delta_2 + q^- - q^+\right|/3 > 0$ it holds that

$$\Delta \geq \frac{\delta_1}{2r}\left|\delta_2 + q^- - q^+\right| - \mathrm{neg}(\kappa) \geq \xi/r.$$

The claim now follows from an averaging argument. □

We are now ready to construct our adversary. Assume without loss of generality that there exists $i \in [r]$ such that

$$\Pr\left[\tilde{a}_i = 1 \wedge \tilde{b}_{i-1} = 1\right] + \Pr\left[\tilde{a}_i \neq 1 \wedge \tilde{b}_i = 1\right] - (\delta_2 + q^-) \geq \xi/r.$$

The case where the expression on the left-hand side is upper bounded by $-\xi/r$ is handled by first observing that

$$\begin{aligned}
&\Pr[\tilde{a}_i = z, \tilde{b}_{i-1} = 1] + \Pr[\tilde{a}_i \neq z, \tilde{b}_i = 1] - (\delta_2 + q^-) \\
&\quad = \Pr[\tilde{a}_i = z] - \Pr[\tilde{a}_i = z, \tilde{b}_{i-1} = 0] + \Pr[\tilde{a}_i \neq z] - \Pr[\tilde{a}_i \neq z, \tilde{b}_i = 1] - (\delta_2 + q^-) \\
&\quad = -(\Pr[\tilde{a}_i = z, \tilde{b}_{i-1} = 0] + \Pr[\tilde{a}_i \neq z, \tilde{b}_i = 0] - (1 - \delta_2 - q^-)),
\end{aligned}$$

and then applying an analogous argument. We define the adversary $\mathcal{A}$ as follows.

1. Corrupt A and C, and instruct them to act honestly until receiving i messages from B.
2. Compute the backup value a_i as an honest A and C would in case B aborts. If $a_i \oplus \mathsf{flip}(x) = 1$ then instruct A to send the next message honestly and then abort.
3. Otherwise, instruct A to abort before sending the next message.
4. In both cases, C act honestly until the end of the protocol, where it obtains a backup value b.
5. Output $b \oplus 1$ as the guess for $\mathsf{flip}(y)$.

Observe that $\mathcal{A}$ guesses $\mathsf{flip}(y)$ with a probability that is significantly greater than $\delta_2 + q^-$. Indeed,

$$\begin{aligned}
\Pr[b \oplus 1 = \mathsf{flip}(y)] &= \Pr[\tilde{a}_i = 1 \wedge b_{i-1} \oplus 1 = \mathsf{flip}(y)] + \Pr[\tilde{a}_i = 0 \wedge b_i \oplus 1 = \mathsf{flip}(y)] \\
&= \Pr\left[\tilde{a}_i = 1 \wedge \tilde{b}_{i-1} = 1\right] + \Pr\left[\tilde{a}_i = 0 \wedge \tilde{b}_i = 1\right] \\
&\geq \delta_2 + q^- + \xi/r.
\end{aligned}$$

□

We now prove Lemma 4.3.

Proof of Lemma 4.3. We only prove the first assertion, as the second is analogous. In the following, for the sake of brevity, we write S instead of S_A, and we fix $x \in \mathcal{X}$ and remove it from S. Additionally, denote $z = f(x, y)$ and $\tilde{z} = z \oplus \mathsf{flip}(y)$. It holds that

$$\begin{aligned}
\Pr[\mathsf{S}(z) = \mathsf{flip}(y)] &= \Pr[\mathsf{S}(z) \oplus z = \tilde{z}] \\
&= \Pr[\mathsf{S}(0) = 0, z = 0, \tilde{z} = 0] + \Pr[\mathsf{S}(1) = 1, z = 1, \tilde{z} = 0] \\
&\quad + \Pr[\mathsf{S}(0) = 1, z = 0, \tilde{z} = 1] + \Pr[\mathsf{S}(1) = 0, z = 1, \tilde{z} = 1] \\
&= \Pr[\mathsf{S}(0) = 0] \cdot \Pr[z = 0, \tilde{z} = 0] + \Pr[\mathsf{S}(1) = 1] \cdot \Pr[z = 1, \tilde{z} = 0] \\
&\quad + \Pr[\mathsf{S}(0) = 1] \cdot \Pr[z = 0, \tilde{z} = 1] + \Pr[\mathsf{S}(1) = 0] \cdot \Pr[z = 1, \tilde{z} = 1] \\
&\leq \max\{\Pr[z = 0, \tilde{z} = 0], \Pr[z = 0, \tilde{z} = 1]\} \\
&\quad + \max\{\Pr[z = 1, \tilde{z} = 0], \Pr[z = 1, \tilde{z} = 1]\}.
\end{aligned}$$

Depending on which quantities are larger, the above expression is upper-bounded by one of the following.

- $\Pr[\tilde{z} = z] = \Pr[\mathsf{flip}(y) = 0] = q^+$,
- $\Pr[\tilde{z} \neq z] = \Pr[\mathsf{flip}(y) = 1] = q^-$,
- $\Pr[\tilde{z} = 0] = -\delta_2 + q^+$,
- $\Pr[\tilde{z} = 1] = \delta_2 + q^-$,

where the last equality in the first two equations is by the definition of q^+ and q^-, and the equality in the last two equations follows from Item 3 of Lemma 4.4. Since $\delta_2 > 0$ and $\delta_2 \geq \mathbf{1}^T \cdot \mathbf{q} = q^+ - q^-$, it follows that the maximum is $\delta_2 + q^-$. □

□

5 A Positive Result for Solitary Output Computation

In this section, we prove our positive results, showing that certain functionalities that are not strong semi-balanced can be computed with full security. Formally, we prove the following.

Lemma 5.1 (Restatement of Lemma 3.6)**.** *Let $f : \mathcal{X} \times \mathcal{Y} \times \{\lambda\} \to \{0,1\}$ be a solitary output three-party Boolean functionality. Assume there exists a locking strategy $\mathbf{p}$ for A satisfying*

$$\mathbf{p}^T \cdot \mathbf{M}_f = \mathbf{1}^T \quad \textit{and} \quad \mathbf{1}^T \cdot \mathbf{p} \geq 1.$$

Then, if a secure protocol for OT exists, f can be computed with full security.

Proof. First, observe that we may assume that $\mathbf{p}$ contains a negative entry, as otherwise f is forced (see Definition 2.5) and can be computed by the results of [26]. We next present the protocol in the dealer model (see Section 2.3). We first introduce some notations. We let $r = \omega(\log(\kappa))$ be the number of rounds, let $p^* = \min_{x \in \mathcal{X}} \{p_x\} < 0$, and we fix two parameters $\beta = (\mathbf{1}^T \cdot \mathbf{p})^{-1}$ and

$$\alpha = \frac{\beta^{-1}}{\beta^{-1} - p^* \cdot |\mathcal{X}|}.$$

Observe that $0 < \beta \leq 1$ by assumption, and that $0 < \alpha < 1$ since $p^* < 0$.

We are now ready to present the protocol. For the sake of presentation, we will describe it assuming that C never aborts, and that A and B cannot both abort. To handle those cases, if C aborts during any part of the protocol, then A and B halt, and if A and B abort then C outputs $f(x_0, y_0)$, where $x_0 \in \mathcal{X}$ and $y_0 \in \mathcal{Y}$ are default values. Our protocol can be seen as a generalization of the two-party protocol by [7]. Roughly, their protocol follows the ideas of the GHKL protocol, however, at round $i^* - 1$ the backup value of B is a constant bit independent of the parties' inputs (though the bit depends on the function). In our three-party protocol, at round $i^* - 1$ the backup value of (B, C) is 1 with probability β.

Protocol 5.2.
Private inputs: Party A holds input $x \in \mathcal{X}$ and party B holds $y \in \mathcal{Y}$. Party C has no private input.
Common input: All parties hold the security parameter 1^κ.

1. *A and B send their inputs to the dealer.*
2. *The dealer sample $i^* \leftarrow \mathsf{Geom}(\alpha)$.*
3. *The dealer computes $w = f(x, y)$ and set $\tilde{b} = 1$ with probability β.*
4. *The dealer computes backup values as follows: For all $i \in \{0, \ldots, r\}$ let*

$$a_i = \begin{cases} f(x, \tilde{y}_i) & \text{if } i < i^* \\ w & \text{otherwise} \end{cases} \quad \text{and} \quad b_i = \begin{cases} f(\tilde{x}_i, y) & \text{if } i < i^* - 1 \\ \tilde{b} & \text{if } i = i^* - 1 \\ w & \text{otherwise} \end{cases}$$

   where $\tilde{x}_i \leftarrow \mathcal{X}$, $\tilde{y}_i \leftarrow \mathcal{Y}$, and $\tilde{b} = 1$ with probability β, are all independent.
5. *The dealer sends b_0 to B and C, held shared in a 2-out-of-2 additive secret sharing scheme.*
6. *For $i = 1$ to r the dealer does the following:*
   (a) *Send a_i to A and C, held shared in a 2-out-of-2 additive secret sharing scheme. If A aborts, then B sends to C it share of b_{i-1} who then outputs it.*
   (b) *Send b_i to B and C, held shared in a 2-out-of-2 additive secret sharing scheme. If B aborts, then A sends to C it's share of a_i who then outputs it.*
7. *A sends its share of a_r to C who then outputs it.*

Observe that correctness holds since $i^* > r$ occurs with only a negligible probability. We next show that the protocol is fully secure. First, observe that corrupting C (and possibly another party) will not provide the adversary with any advantage. Intuitively, this is due to the fact the adversary can obtain at most one application of f, applied to the honest party's input. We defer the formal arguments to Appendix B.1.

One subtlety that we need to consider is the last case, where exactly one party among A and B is corrupted. This is due to the fact that, although the view consists of random shares, a single corrupted party might affect the correctness of the output of C. Note that a corrupt B can be handled easily, since the backup value of (A, C) at every round is of the form $f(x, y')$, for $y' \in \mathcal{Y}$ that is either uniformly random, or equal to the real input y sent to the dealer. Thus, a simulator can send according to the correct distribution by checking whether the adversary aborts before or after round i^*.

We now consider the case where A is corrupted. Unlike the previous case, the output of C in round $i^* - 1$ is not of the form $f(x', y)$ for any (even possibly random) $x' \in \mathcal{X}$. Thus, the naive simulation from before does not work. Nevertheless, we next show that there exists a distribution over $\mathcal{X}$, such that the output of C in the ideal world is identical to the output of C in the real world. We defer the formal description of the simulator to Appendix B.1. We next analyze the distribution of the output $\textsc{out}$ of C in the real world and compare it to an ideal execution.

In the following, all probabilities are conditioned on the adversary aborting at round i and on $i^* \leq r$. Let $\mathbf{e}_x \in \mathbb{R}^{|\mathcal{X}|}$ denote the x^{th} standard basis vector. Observe that in the real world, it holds that

$$\begin{aligned}\Pr[\textsc{out} = 1] &= \Pr[i < i^*] \cdot \Pr[f(\tilde{x}_i, y) = 1] + \Pr[i = i^*] \cdot \beta + \Pr[i > i^*] \cdot f(x, y) \\ &= (1-\alpha)^i \cdot \mathbf{u}_{\mathcal{X}} \cdot \mathbf{M}_f(\cdot, y) + (1-\alpha)^{i-1} \cdot \alpha\beta + \left(1 - (1-\alpha)^{i-1}\right) \cdot \mathbf{e}_x \cdot \mathbf{M}_f(\cdot, y), \qquad (1)\end{aligned}$$

where $\mathbf{u}_{\mathcal{X}}$ is the uniform probability (row) vector over $\mathcal{X}$. On the other hand, in the ideal world, the simulator sends a value according to a distribution that depends only on the input x and the round i in which the adversary aborted. We denote this distribution by the row vector $\mathbf{x}_{x,i}$. As the ideal world is identically distributed to the real world, it follows that for all $y \in \mathcal{Y}$ it holds that

$$\mathbf{x}_{x,i} \cdot \mathbf{M}_f(\cdot, y) = (1-\alpha)^i \cdot \mathbf{u}_{\mathcal{X}} \cdot \mathbf{M}_f(\cdot, y) + (1-\alpha)^{i-1} \cdot \alpha\beta + \left(1 - (1-\alpha)^{i-1}\right) \cdot \mathbf{e}_x^T \cdot \mathbf{M}_f(\cdot, y).$$

Since this must hold for all $y \in \mathcal{Y}$, we may write

$$\mathbf{x}_{x,i} \cdot \mathbf{M}_f = (1-\alpha)^i \cdot \mathbf{u}_{\mathcal{X}} \cdot \mathbf{M}_f + (1-\alpha)^{i-1} \cdot \alpha\beta \cdot \mathbf{1}^T + \left(1 - (1-\alpha)^{i-1}\right) \cdot \mathbf{e}_x^T \cdot \mathbf{M}_f. \qquad (2)$$

We now show that Equation (2) admits a solution $\mathbf{x}_{x,i}$ that is also a probability vector. Since $\mathbf{1}^T = \mathbf{p}^T \cdot \mathbf{M}_f$, we may write the right-hand side as

$$\left((1-\alpha)^i \cdot \mathbf{u}_{\mathcal{X}} + (1-\alpha)^{i-1} \cdot \alpha\beta \cdot \mathbf{p}^T + \left(1 - (1-\alpha)^{i-1}\right) \cdot \mathbf{e}_x^T\right) \cdot \mathbf{M}_f.$$

Thus, the row vector $\mathbf{v} := (1-\alpha)^i \cdot \mathbf{u}_{\mathcal{X}} + (1-\alpha)^{i-1} \cdot \alpha\beta \cdot \mathbf{p}^T + \left(1 - (1-\alpha)^{i-1}\right) \cdot \mathbf{e}_x^T$ solves Equation (2). To conclude the proof, we next show that $\mathbf{v}$ is a probability vector, i.e., its entries are non-negative and sum to 1. First, observe that

$$\begin{aligned}\mathbf{v} \cdot \mathbf{1} &= (1-\alpha)^i + (1-\alpha)^{i-1} \cdot \alpha\beta \cdot \left(\mathbf{p}^T \cdot \mathbf{1}\right) + 1 - (1-\alpha)^{i-1} \\ &= (1-\alpha)^i + (1-\alpha)^{i-1} \cdot \alpha + 1 - (1-\alpha)^{i-1} \\ &= 1.\end{aligned}$$

Second, for every $x' \in \mathcal{X}$, it holds that

$$\begin{aligned}v(x') &\geq (1-\alpha)^i \cdot \frac{1}{|\mathcal{X}|} + (1-\alpha)^{i-1} \cdot \alpha\beta \cdot p_{x'} \\ &= (1-\alpha)^{i-1} \cdot \left(\frac{1-\alpha}{|\mathcal{X}|} + \alpha\beta \cdot p_{x'}\right) \\ &= (1-\alpha)^{i-1} \cdot \frac{1 - \alpha\left(1 - \beta \cdot p_{x'} \cdot |\mathcal{X}|\right)}{|\mathcal{X}|} \\ &= (1-\alpha)^{i-1} \cdot \frac{1 - \frac{\beta^{-1}}{\beta^{-1} - p^* \cdot |\mathcal{X}|}\left(1 - \beta \cdot p_{x'} \cdot |\mathcal{X}|\right)}{|\mathcal{X}|} \\ &= (1-\alpha)^{i-1} \cdot \frac{p_{x'} - p^*}{\beta^{-1} - p^* \cdot |\mathcal{X}|} \\ &\geq 0.\end{aligned}$$

Thus, $\mathbf{v}$ is a probability vector as required. □

6 Application: Analysis of the Disjointness Functionality

In this section, we use Theorem 3.2 to analyze the parameters for which the disjointness functionality can be computed with full security. Throughout the section, for natural numbers $k, m, n \in \mathbb{N}$ satisfying $k \leq m \leq n$, we denote

$$\binom{[n]}{k, m} = \{\mathcal{S} \subseteq [n] : k \leq |\mathcal{S}| \leq m\}.$$

We prove the following.

Theorem 6.1 (Restatement of Theorem 3.7)**.** *Let $n, k \in \mathbb{N}$, where $1 \leq k < n/2$. Define the solitary output three-party Boolean functionality $\mathsf{disj} : \binom{[n]}{k,n-k}^2 \times \{\lambda\} \to \{0,1\}$ as*

$$\mathsf{disj}(\mathcal{S}, \mathcal{T}) = \begin{cases} 1 & \text{if } \mathcal{S} \cap \mathcal{T} = \emptyset \\ 0 & \text{otherwise} \end{cases}$$

Then, if a secure protocol for OT exists, disj can be computed with full security if and only if n is even.

We will make use of the following two lemmas.

Lemma 6.2. *Let $n \in \mathbb{N}$ and let $k \in [n]$. Then*

$$\sum_{m=k}^{n} (-1)^{m+k} \cdot \binom{n}{m}\binom{m-1}{k-1} = 1.$$

Lemma 6.3. *Let $n, k \in \mathbb{N}$ be such that $1 \leq k < n/2$. Then*

$$\sum_{m=k}^{n-k} (-1)^{m+k} \cdot \binom{n}{m}\binom{m-1}{k-1} < 1$$

if n is odd, and

$$\sum_{m=k}^{n-k} (-1)^{m+k} \cdot \binom{n}{m}\binom{m-1}{k-1} > 1$$

if n is even.

We prove Lemmas 6.2 and 6.3 below. We first show that they imply Theorem 6.1. Throughout the following section, all summations over sets are restricted to summations over $\binom{[n]}{k,n-k}$.

Proof of Theorem 6.1. We first show that A and B have locking strategies, regardless of the parity of n. We define the vector $\mathbf{p}$ over $\binom{[n]}{k,n-k}$ as $p_{\mathcal{S}} = (-1)^{|\mathcal{S}|+k} \cdot \binom{|\mathcal{S}|-1}{k-1}$. Then for any subset $\mathcal{T} \in \binom{[n]}{k,n-k}$

it holds that

$$\begin{aligned}
\mathbf{p}^T \cdot \mathbf{M}_{\mathsf{disj}}(\cdot, \mathcal{T}) &= \sum_{\mathcal{S}:\mathcal{S}\cap\mathcal{T}=\emptyset} p_{\mathcal{S}} \\
&= \sum_{\mathcal{S}:\mathcal{S}\cap\mathcal{T}=\emptyset} (-1)^{|\mathcal{S}|+k} \cdot \binom{|\mathcal{S}|-1}{k-1} \\
&= \sum_{m=k}^{n-|\mathcal{T}|} \sum_{\substack{\mathcal{S}:|\mathcal{S}|=m \\ \mathcal{S}\cap\mathcal{T}=\emptyset}} (-1)^{m+k} \cdot \binom{m-1}{k-1} \\
&= \sum_{m=k}^{n-|\mathcal{T}|} (-1)^{m+k} \cdot \binom{m-1}{k-1}\binom{n-|\mathcal{T}|}{m} \\
&= 1,
\end{aligned}$$

where the last equality follows from Lemma 6.2. Since disj is symmetric with respect to the input (i.e., $\mathsf{disj}(\mathcal{S}, \mathcal{T}) = \mathsf{disj}(\mathcal{T}, \mathcal{S})$), we define the locking strategy $\mathbf{q}$ for B exactly the same. Then by Theorem 3.2, disj can be computed with full security if and only if $\mathbf{1}^T \cdot \mathbf{p} \geq 1$. By Lemma 6.3, this occurs if and only if n is even.

□

It is left to prove Lemmas 6.2 and 6.3. We will use the following well-known combinatorial identities.

Proposition 6.4. *For all $k, m, n \in \mathbb{N}$ where $k \leq m \leq n$ it holds that*

$$\binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-m}{m-k}.$$

Proposition 6.5. *For all $k, m, n \in \mathbb{N}$, where $1 \leq k \leq m \leq n$ it holds that*

$$\sum_{i=k}^{m} (-1)^i \cdot \binom{n}{i} = (-1)^m \cdot \binom{n-1}{m} - (-1)^{k-1} \cdot \binom{n-1}{k-1}$$

The idea in both the proof of Lemma 6.2 and of Lemma 6.3, is to view the summation as a function $s(k)$, and use Pascal's identity to compute the difference $s(k) - s(k+1)$.

Proof of Lemma 6.2. For every $k \in [n]$ let

$$s(k) = \sum_{m=k}^{n} (-1)^{m+k} \cdot \binom{n}{m}\binom{m-1}{k-1}.$$

Then

$$\begin{aligned}
s(k) - s(k+1) &= \sum_{m=k}^{n} (-1)^{m+k} \cdot \binom{n}{m}\binom{m-1}{k-1} - \sum_{m=k+1}^{n} (-1)^{m+k+1} \cdot \binom{n}{m}\binom{m-1}{k} \\
&= \binom{n}{k} + \sum_{m=k+1}^{n} (-1)^{m+k} \cdot \binom{n}{m} \cdot \left[\binom{m-1}{k-1} + \binom{m-1}{k}\right] \\
&= \binom{n}{k} + \sum_{m=k+1}^{n} (-1)^{m+k} \cdot \binom{n}{m}\binom{m}{k} \\
&= \binom{n}{k} + \sum_{m=k+1}^{n} (-1)^{m+k} \cdot \binom{n}{k}\binom{n-k}{m-k} \\
&= \binom{n}{k} \cdot \left[1 + \sum_{m=k+1}^{n} (-1)^{m+k} \cdot \binom{n-k}{m-k}\right] \\
&= \binom{n}{k} \cdot \left[1 + \sum_{m=1}^{n-k} (-1)^{m} \cdot \binom{n-k}{m}\right] \\
&= \binom{n}{k} \cdot \sum_{m=0}^{n-k} (-1)^{m} \cdot \binom{n-k}{m} \\
&= 0,
\end{aligned}$$

where the third equality is by Pascal's identity, the fourth is by Proposition 6.4, and the last is due to the fact that the alternating sum of all binomial coefficients is 0. The proof now follows from the observation that $s(n) = 1$. $\square$

Proof of Lemma 6.3. For every $k \in \mathbb{N}$ where $1 \le k < n/2$, let

$$s(k) = \sum_{m=k}^{n-k} (-1)^{m+k} \cdot \binom{n}{m}\binom{m-1}{k-1}.$$

Let us first handle the case where n is odd. Here we may write

$$s(k) = \sum_{m=k}^{\frac{n-1}{2}} (-1)^{m+k} \cdot \binom{n}{m}\binom{m-1}{k-1} + \sum_{m=\frac{n+1}{2}}^{n-k} (-1)^{m+k} \cdot \binom{n}{m}\binom{m-1}{k-1}. \tag{3}$$

Continuing with the second term, we have that

$$\begin{aligned}
\sum_{m=\frac{n+1}{2}}^{n-k} (-1)^{m+k} \cdot \binom{n}{m}\binom{m-1}{k-1} &= \sum_{m=k}^{\frac{n-1}{2}} (-1)^{n-m+k} \cdot \binom{n}{n-m}\binom{n-m-1}{k-1} \\
&= \sum_{m=k}^{\frac{n-1}{2}} (-1)^{m+k-1} \cdot \binom{n}{m}\binom{n-m-1}{k-1},
\end{aligned}$$

where the first equality is by substituting m for $n-m$ and the second is due to the fact that n is odd. Plugging into Equation (3) we obtain

$$s(k) = \sum_{m=k}^{\frac{n-1}{2}} (-1)^{m+k} \cdot \binom{n}{m} \cdot \left[\binom{m-1}{k-1} - \binom{n-m-1}{k-1}\right].$$

The lemma follows from the next claim, asserting that $s(k)$ is a bitonic sequence, i.e., it first decreases and then increases.

Proposition 6.6. *It holds that $s(k) \geq s(k+1)$ if and only if $k \leq n/3$.*

The proposition is proven below. We first show that it immediately concludes the proof for this case. Indeed, by Proposition 6.6 it is enough to show that $s(1) < 1$ and $s((n-1)/2) < 1$. Indeed, for $k=1$ it holds that

$$s(1) = \sum_{m=1}^{n-1} (-1)^{m+1} \cdot \binom{n}{m} = -\left[\sum_{m=0}^{n} (-1)^m \cdot \binom{n}{m} - 1 - (-1)^n\right] = 1 + (-1)^n = 0,$$

where the last equality is since n is odd. Similarly, when $k = (n-1)/2$ it holds that

$$s\left(\frac{n-1}{2}\right) = \sum_{m=\frac{n-1}{2}}^{\frac{n+1}{2}} (-1)^{m+\frac{n-1}{2}} \cdot \binom{n}{m}\binom{m-1}{\frac{n-3}{2}} = \binom{n}{\frac{n-1}{2}} - \binom{n}{\frac{n+1}{2}}\binom{\frac{n-1}{2}}{\frac{n-3}{2}} = \binom{n}{\frac{n-1}{2}} \cdot \frac{3-n}{2}.$$

Recall that we assume that $1 < n/2$. Therefore $n \geq 3$, hence $s((n-1)/2) \leq 0$.

We next handle the case where n is even. A similar analysis to the previous case yields that

$$s(k) = (-1)^{\frac{n}{2}+k} \cdot \binom{n}{n/2}\binom{\frac{n}{2}-1}{k-1} + \sum_{m=k}^{\frac{n}{2}-1} (-1)^{m+k} \cdot \binom{n}{m} \cdot \left[\binom{m-1}{k-1} + \binom{n-m-1}{k-1}\right].$$

Similarly to Proposition 6.6, $s(k)$ is bitonic in this case as well (although it starts increasing and then decreasing).

Proposition 6.7. *It holds that $s(k) \leq s(k+1)$ if and only if $k \leq n/3$.*

As in the previous case, we are left with verifying the inequality for $k=1$ and $k=n/2$. For the former case, it holds that

$$s(1) = \sum_{m=1}^{n-1} (-1)^{m+1} \cdot \binom{n}{m} = -\left[\sum_{m=0}^{n} (-1)^m \cdot \binom{n}{m} - 1 - (-1)^n\right] = 1 + (-1)^n = 2,$$

and for the latter case, it holds that $s(n/2) = \binom{n}{n/2}$. $\square$

It is left to prove Propositions 6.6 and 6.7. We next prove Proposition 6.6. The proof of Proposition 6.7 follows a similar analysis and is provided in Appendix B.2 for completeness.

Proof of Proposition 6.6. We compute the difference $s(k)-s(k+1)$ and express it in simpler terms. It holds that

$$\begin{aligned}
s(k)-s(k+1) &= \sum_{m=k}^{\frac{n-1}{2}} (-1)^{m+k} \cdot \binom{n}{m} \cdot \left[\binom{m-1}{k-1} - \binom{n-m-1}{k-1}\right] \\
&\quad - \sum_{m=k+1}^{\frac{n-1}{2}} (-1)^{m+k+1} \cdot \binom{n}{m} \cdot \left[\binom{m-1}{k} - \binom{n-m-1}{k}\right] \\
&= \binom{n}{k} \cdot \left[1 - \binom{n-k-1}{k-1}\right] \\
&\quad + \sum_{m=k+1}^{\frac{n-1}{2}} (-1)^{m+k} \cdot \binom{n}{m} \cdot \left[\binom{m-1}{k-1} + \binom{m-1}{k} - \binom{n-m-1}{k-1} - \binom{n-m-1}{k}\right] \\
&= \binom{n}{k} \cdot \left[1 - \binom{n-k-1}{k-1}\right] + \sum_{m=k+1}^{\frac{n-1}{2}} (-1)^{m+k} \cdot \binom{n}{m} \cdot \left[\binom{m}{k} - \binom{n-m}{k}\right],
\end{aligned}$$

where the last equality is by Pascal's identity. By Proposition 6.4, it holds that $\binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-k}{m-k}$ and that $\binom{n}{m}\binom{n-m}{k} = \binom{n}{m+k}\binom{m+k}{k} = \binom{n}{k}\binom{n-k}{m}$. Therefore,

$$s(k)-s(k+1) = \binom{n}{k} \cdot \left(1 - \binom{n-k-1}{k-1} + \sum_{m=k+1}^{\frac{n-1}{2}} (-1)^{m+k} \cdot \left[\binom{n-k}{m-k} - \binom{n-k}{m}\right]\right). \qquad (4)$$

Now, continuing the summation of the first term we obtain

$$\sum_{m=k+1}^{\frac{n-1}{2}} (-1)^{m+k} \cdot \binom{n-k}{m-k} = \sum_{m=1}^{\frac{n-1}{2}-k} (-1)^{m} \cdot \binom{n-k}{m} = (-1)^{\frac{n-1}{2}-k} \cdot \binom{n-k-1}{\frac{n-1}{2}-k} - 1,$$

where last equality is by Proposition 6.5. Similarly, the summation of the second term equals to

$$\sum_{m=k+1}^{\frac{n-1}{2}} (-1)^{m+k} \cdot \binom{n-k}{m} = (-1)^{\frac{n-1}{2}+k} \cdot \binom{n-k-1}{\frac{n-1}{2}} - \binom{n-k-1}{k}.$$

Plugging it back into Equation (4) we obtain

$$\begin{aligned}
s(k) - s(k+1) &= \binom{n}{k} \cdot \left(1 - \binom{n-k-1}{k-1} + (-1)^{\frac{n-1}{2}-k} \cdot \binom{n-k-1}{\frac{n-1}{2}-k} - 1 \right. \\
&\qquad \left. - \left[(-1)^{\frac{n-1}{2}-k} \cdot \binom{n-k-1}{\frac{n-1}{2}} - \binom{n-k-1}{k}\right]\right) \\
&= \binom{n}{k} \cdot \left(\binom{n-k-1}{k} - \binom{n-k-1}{k-1} \right. \\
&\qquad \left. + (-1)^{\frac{n-1}{2}-k} \cdot \left[\binom{n-k-1}{\frac{n-1}{2}-k} - \binom{n-k-1}{\frac{n-1}{2}}\right]\right) \\
&= \binom{n}{k} \cdot \left[\binom{n-k-1}{k} - \binom{n-k-1}{k-1}\right] \\
&= \binom{n}{k}\binom{n-k-1}{k-1} \cdot \frac{n-3k}{k},
\end{aligned}$$

which is non-negative if and only if $k \leq n/3$. □

7 The Multiparty Setting

In this section, we consider the multiparty setting and show a positive result. We denote the number of parties by n, and denote their identities by $\mathsf{P}_0, \ldots, \mathsf{P}_n$. The parties compute an $(n+1)$-party solitary output (not necessarily Boolean) functionality $f : (\{0,1\}^*)^{n+1} \to \{0,1\}^*$, where the output receiving party is P_0.

7.1 Extending the Security Definition

The security definition of a protocol is naturally extended using the real/ideal paradigm. Given a protocol π, an adversary $\mathcal{A}$, a tuple of inputs $\mathbf{x} \in (\{0,1\}^*)^{n+1}$, and an auxiliary input $\mathsf{aux} \in \{0,1\}^*$, we let $\text{REAL}_{\pi,\mathcal{A}(\mathsf{aux})}(\kappa, \mathbf{x})$ denote the view of the adversary $\mathcal{A}$ and the output of (an honest) P_0 in an execution of π on inputs $\mathbf{x}$, and security parameter κ interacting with $\mathcal{A}$ with auxiliary input aux. We further let $\text{IDEAL}_{f,\mathcal{A}(\mathsf{aux})}(\kappa, \mathbf{x})$ denote the joint view of $\mathcal{A}$ being its output in a random execution of the ideal-world process, and the output (an honest) P_0. The security definition is as follows.

Definition 7.1 (Malicious security for the multiparty setting)**.** *Let $n, t \in \mathbb{N}$, where $t \leq n$, let $f : (\{0,1\}^*)^{n+1} \to \{0,1\}^*$ be an $(n+1)$-party functionality, and let π be an $(n+1)$-party protocol. We say that π computes f with t* full security*, if for every non-uniform* PPT *adversary $\mathcal{A}$, controlling a subset $\mathcal{I} \subseteq \{\mathsf{P}_0, \ldots, \mathsf{P}_n\}$ of size at most t in the real world, there exists a non-uniform* PPT *adversary* Sim*, controlling the same subset $\mathcal{I}$ in the ideal-world such that*

$$\left\{\text{IDEAL}_{f,\mathsf{Sim}(\mathsf{aux})}(\kappa, \mathbf{x})\right\}_{\kappa\in\mathbb{N},\mathbf{x}\in(\{0,1\}^*)^{n+1},\mathsf{aux}\in\{0,1\}^*} \overset{\text{C}}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(\mathsf{aux})}(\kappa, \mathbf{x})\right\}_{\kappa\in\mathbb{N},\mathbf{x}\in(\{0,1\}^*)^{n+1},\mathsf{aux}\in\{0,1\}^*}.$$

Ideal computation with security-with-identifiable-abort. In this section, we also use a security notion called security-with-identifiable-abort where the adversary can cause the computation to prematurely abort. However, it can do so after learning the output, at the expense of revealing the identity of a corrupted party (see Appendix A for a formal definition). We say that a protocol π computes a functionality f with t-security-with-identifiable-abort if any adversary can be simulated in the secure-with-identifiable-abort ideal model, similarly to Definition 7.1.

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.

Let f be a functionality. Then, an execution of a protocol π computing a functionality g in the $(f, \mathsf{id\text{-}abort})$-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 with security-with-identifiable-abort. 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.

The sequential composition theorem of Canetti [13] states the following. Let ρ be a protocol that securely computes f in the ideal model $\mathsf{id\text{-}abort}$. Then, if a protocol π computes g in the $(f, \mathsf{id\text{-}abort})$-hybrid model, then the protocol π^ρ, that is obtained from π by replacing all ideal calls to the trusted party computing f with the protocol ρ, securely computes g in the real model.

Theorem 7.2 ([13]). *Let $n, t \in \mathbb{N}$, where $t \leq n$, let $f : (\{0,1\}^*)^{n+1} \to \{0,1\}^*$ be an $(n+1)$-party functionality, let ρ be a protocol that computes f with t-security-with-identifiable-abort, and let π be a protocol that t-securely computes g with t full security in the $(f, \mathsf{id\text{-}abort})$-hybrid model. Then, protocol π^ρ computes g with t-full security in the real model.*

7.2 Our Results

We construct an $(n-t+1)$-secure protocol for functionalities with the following property: Any t parties among $\{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ (i.e., those that do not receive the output) can fix the output distribution using an appropriate distribution over their inputs. We refer to these functionalities as t-forced. Note that it suffices to consider the case where $t \leq (n+1)/2$ since otherwise, $n-t+1 < (n+1)/2$ implies there is an honest majority.

Definition 7.3. *Let $n, t \in \mathbb{N}$, where $t \leq (n+1)/2$, and let $f : (\{0,1\}^*)^{n+1} \to \{0,1\}^*$ be an $(n+1)$-party solitary output functionality. We say that f is* t-forced *if there exists a distribution D over the set of outputs such that the following holds. For any set $\mathcal{S} \subseteq \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ of size t there exists a distribution $D_{\mathcal{S}}$ over their inputs that fixes the output distribution of f to be D. That is, for any $x_i \in \{0,1\}^*$, where $i \notin \mathcal{S}$, it holds that $f(x_0, \ldots, x_n)$, where $(x_j)_{j \in \mathcal{S}} \leftarrow D_{\mathcal{S}}$, is identically distributed to a sample from D.*

We are now ready to state our result, showing that any t-forced solitary output functionality can be securely computed against $n-t+1$ corruptions.

Theorem 7.4. *Let $n, t \in \mathbb{N}$, where $t \leq (n+1)/2$, and let $f : (\{0,1\}^*)^{n+1} \to \{0,1\}^*$ be an $(n+1)$-party solitary output t-forced functionality. Then, assuming oblivious transfer exists, f can be computed with $(n-t+1)$ full security.*

Proof. We show a protocol that is secure against fail-stop adversaries, i.e., adversaries that follow the specifications of the protocol, however, they may abort parties prematurely. A protocol secure against any malicious adversary can be obtained using standard techniques.

Let ShrGen denote the following $(n+1)$-party functionality: Each party P_i sends its inputs x_i, the functionality computes $(n-t+2)$-out-of-$(n+1)$ Shamir's secret sharing of the output $z = f(x_0, \ldots, x_n)$, and for every $i \in \{0, \ldots, n\}$ it gives the i^{th} share, denoted $z[i]$, to P_i. The protocol for computing f is as follows.

Protocol 7.5. *Private inputs: For every $i \in \{0, \ldots, n\}$ party P_i holds input x_i.*
Common input: All parties hold the security parameter 1^κ.

1. *The parties call $(\mathsf{ShrGen}, \mathsf{id\text{-}abort})$ using their inputs. In case of abort:*
   - *If the parties receive $(\mathsf{abort}, \mathsf{P}_0)$, then all parties halt.*
   - *If the parties receive $(\mathsf{abort}, \mathsf{P})$ for some party $\mathsf{P} \neq \mathsf{P}_0$, then the parties restart with the input of P set to a default value.*
2. *If no abort occurred, each active party P_i receives a share $z[i]$.*
3. *Each active party sends its share to P_0.*
4. *If P_0 received at least $n-t+2$ shares, then it reconstructs the output and halt.*
5. *Otherwise, P_0 outputs a sample from D.*

Clearly, the protocol is correct. First, consider an adversary $\mathcal{A}$ corrupting the output receiving party P_0, and at most $n-t$ of the parties in $\{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$. Its simulator is defined as follows.

1. Query the adversary for its inputs to $(\mathsf{ShrGen}, \mathsf{id\text{-}abort})$.
2. Give the adversary random shares as its output from the call to $(\mathsf{ShrGen}, \mathsf{id\text{-}abort})$.
3. If the adversary replies with $(\mathsf{abort}, \mathsf{P})$ then do the following:
   - If $\mathsf{P} = \mathsf{P}_0$, then output whatever $\mathcal{A}$ outputs, and halt.
   - Otherwise, if $\mathsf{P} \neq \mathsf{P}_0$, then go back to Step 1.
4. Otherwise, if $\mathcal{A}$ sent $\mathsf{continue}$, send to the trusted party, the inputs chosen for the corrupted parties at Step 1 (replacing the inputs of aborting parties with default values).
5. Complete the Shamir's shares previously given to $\mathcal{A}$ to shares for the honest parties (note that this can be done by the properties of Shamir's secret sharing scheme).
6. Send to $\mathcal{A}$ the shares generated for the honest parties, output whatever $\mathcal{A}$ outputs, and halt.

By the properties of the secret sharing scheme, since $\mathcal{A}$ does not hold enough shares, it will behave in the ideal world exactly the same as in the real world. In particular, it will obtain the same output of f.

Now, consider an adversary $\mathcal{A}$ that *does not* corrupt the output receiving party P_0, and corrupts a set $\mathcal{I} \subseteq \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ of at most $n - t + 1$ parties. Its simulator is defined as follows.

1. Query the adversary for its inputs to $(\mathsf{ShrGen}, \mathsf{id\text{-}abort})$.
2. Give the adversary random shares as its output from the call to $(\mathsf{ShrGen}, \mathsf{id\text{-}abort})$.
3. If the adversary replies with $(\mathsf{abort}, \mathsf{P})$ then go back to Step 1.
4. Otherwise, if $\mathcal{A}$ sent $\mathsf{continue}$, query it for the shares the corrupted parties send to P_0.
5. If $\mathcal{A}$ sent aborted at least t parties aborted (overall, including the calls to ShrGen), then samples $(x_i)_{i \in \mathcal{I}} \leftarrow D_{\mathcal{I}}$, and send it to the trusted party.
6. Otherwise, send to the trusted party the inputs chosen for the corrupted parties at Step 1 (replacing the inputs of aborting parties with default values).
7. Output whatever $\mathcal{A}$ outputs and halt.

Similarly to the previous case, since $\mathcal{A}$ does not hold enough shares, it will behave in the ideal world exactly the same as in the real world. By the assumption that the function is t-forced, if the adversary aborted at least t parties then P_0 will output a sample from D in both worlds. Otherwise, in both worlds, it will output $f(x_0, \ldots, x_n)$, where each x_i is either an input of an honest party, or the input chosen by $\mathcal{A}$ for the corrupted party, or a default value for a corrupted party that aborted. $\square$

Bibliography

[1] N. Agarwal, S. Anand, and M. Prabhakaran. Uncovering algebraic structures in the mpc landscape. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 381–406. Springer, 2019.

[2] S. Agrawal and M. Prabhakaran. On fair exchange, fair coins and fair sampling. In *CRYPTO*, pages 259–276, 2013.

[3] B. Alon and E. Omri. On secure computation of solitary output functionalities with and without broadcast. *Cryptology ePrint Archive*, 2022.

[4] B. Alon, R. Cohen, E. Omri, and T. Suad. On the power of an honest majority in three-party computation without broadcast. In *Theory of Cryptography Conference*, pages 621–651. Springer, 2020.

[5] B. Alon, M. Naor, E. Omri, and U. Stemmer. Mpc for tech giants (gmpc): Enabling gulliver and the lilliputians to cooperate amicably. *arXiv preprint arXiv:2207.05047*, 2022.

[6] G. Asharov. Towards characterizing complete fairness in secure two-party computation. In *TCC*, pages 291–316, 2014.

[7] G. Asharov, A. Beimel, N. Makriyannis, and E. Omri. Complete characterization of fairness in secure two-party computation of Boolean functions. In *Proceedings of the 12th Theory of Cryptography Conference(TCC), part I*, pages 199–228, 2015.

[8] A. Beimel, A. Gabizon, Y. Ishai, E. Kushilevitz, S. Meldgaard, and A. Paskin-Cherniavsky. Non-interactive secure multiparty computation. In *Annual Cryptology Conference*, pages 387–404. Springer, 2014.

[9] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1253–1269, 2020.

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

[11] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[12] L. Burkhalter, H. Lycklama, A. Viand, N. Küchler, and A. Hithnawi. Rofl: Attestable robustness for secure federated learning. *arXiv preprint arXiv:2107.03311*, 2021.

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

[14] D. Chaum, C. Crépeau, and I. Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19, 1988. doi: 10.1145/62212.62214.

[15] R. Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *Proceedings of the 18th Annual ACM STOC*, 1986.

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

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

[18] D. Dachman-Soled. Revisiting fairness in MPC: polynomial number of parties and general adversarial structures. In R. Pass and K. Pietrzak, editors, *Proceedings of the 18th Theory of Cryptography Conference(TCC), part II*, volume 12551, pages 595–620. Springer, 2020.

[19] V. Daza and N. Makriyannis. Designing fully secure protocols for secure two-party computation of constant-domain functions. In *Proceedings of the 15th Theory of Cryptography Conference(TCC), part I*, pages 581–611, 2017.

[20] U. Feige, J. Killian, and M. Naor. A minimal model for secure computation. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 554–563, 1994.

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

[22] 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 51st Annual ACM STOC*, pages 218–229, 1987.

[23] S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In *Conference on the Theory and Application of Cryptography*, pages 77–93. Springer, 1990.

[24] S. 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)*, pages 19–35, 2009.

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

[26] S. Halevi, Y. Ishai, E. Kushilevitz, N. Makriyannis, and T. Rabin. On fully secure MPC with solitary output. In *Proceedings of the 17th Theory of Cryptography Conference(TCC), part I*, pages 312–340, 2019.

[27] Y. Ishai, E. Kushilevitz, Y. Lindell, and E. Petrank. On combining privacy with guaranteed output delivery in secure multiparty computation. In *Annual International Cryptology Conference*, pages 483–500. Springer, 2006.

[28] 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.

[29] J. Katz. On achieving the" best of both worlds" in secure multiparty computation. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 11–20, 2007.

[30] 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), part I*, pages 565–580, 2017.

[31] 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)*, pages 135–154, 2014.

[32] N. Makriyannis et al. *Fairness in two-party computation: characterizing fair functions.* PhD thesis, Universitat Pompeu Fabra, 2016.

[33] 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.

[34] A. C. Yao. Protocols for secure computations (extended abstract). In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 160–164, 1982.

A Definition of Security-With-Identifiable-Abort

We next define an ideal computation with *security-with-identifiable-abort*, where a trusted party performs the computation on behalf of the parties, and where the ideal-model adversary can abort the computation after learning the output, but at the expense of revealing the identity of a corrupted party. An ideal computation of a (not necessarily solitary output) $(n+1)$-party functionality f, on an input tuple $\mathbf{x} = (x_0, \ldots, x_n) \in (\{0,1\}^*)^{n+1}$ and security parameter κ, with an ideal-world adversary $\mathcal{A}$ running with an auxiliary input aux and corrupting a subset $\mathcal{I}$ of the parties proceeds as follows:

Parties send inputs to the trusted party: Each honest party P_i sends its input x_i to the trusted party. For each corrupted party P_i, the adversary $\mathcal{A}$ sends a value x_i^*. If no value was sent, then the trusted party replaces it with a default value. Let $\mathbf{x}' = (x'_0, \ldots, x'_n)$ denote the inputs received by the trusted party (possibly after replacing with default values).

The trusted party performs computation: The trusted party selects a random string r, computes $(z_0, \ldots, z_n) = f(\mathbf{x}'; r)$, and sends $(z_i)_{i:\mathsf{P}_i \in \mathcal{I}}$ to $\mathcal{A}$.

Malicious adversary instructs trusted party to continue or halt: The adversary $\mathcal{A}$ sends either $\mathsf{continue}$ or $(\mathsf{abort}, \mathsf{P})$ for some $\mathsf{P} \in \mathcal{I}$ to T. If it sent $\mathsf{continue}$, then for every honest party P_i the trusted party sends it z_i. Otherwise, if $\mathcal{A}$ sent $(\mathsf{abort}, \mathsf{P})$, then T sends $(\mathsf{abort}, \mathsf{P})$ to the all honest parties.

Outputs: Each honest party outputs whatever output it received from the trusted party and the corrupted parties output nothing. The adversary $\mathcal{A}$ outputs some function of its view (i.e., the auxiliary input, its randomness, and the input and output of the corrupted parties).

B Missing Proofs

B.1 Formal simulators for Protocol 5.2

We next present the formal descriptions of the simulators.

B.1.1 Simulator for a corrupt A

The simulator for a corrupt A is described as follows.

1. Query the adversary for the input x it sends to the dealer.
2. Sample $i^* \leftarrow \mathsf{Geom}(\alpha)$.
3. For $i = 1$ to $i^* - 1$:
   (a) Send to $\mathcal{A}$ a random bit, representing the share of the backup value a_i.
   (b) If $\mathcal{A}$ aborts, then send to the trusted party T a random input $x^* \leftarrow \mathbf{x}_{x,i}$, where $\mathbf{x}_{x,i}$ is defined as the solution to equation Equation (2). Output the random shares as the adversary's view.
4. If $\mathcal{A}$ did not abort A, send x to T and obtain the output $w = f(x, y)$.
5. For $i = i^*$ to r:
   (a) Send to $\mathcal{A}$ a random bit, representing the share of the backup value $a_i = w$.
   (b) If $\mathcal{A}$ aborts, output the random shares as the adversary's view.
6. If no abort occurred, output the random shares as the adversary's view.

B.1.2 Simulators for when C is corrupted

Observe that if only C is corrupted, then security is trivial as it receives only random shares during the entire interaction, and no other party obtains the output. We now assume that the adversary $\mathcal{A}$ corrupts both A and C. The case where B is corrupted is similar and is done below. Define the simulator $\mathsf{Sim}_{\mathcal{A}}$ as follows.

1. Query the adversary for the input x it sends to the dealer.

2. Sample $i^* \leftarrow \mathsf{Geom}(\alpha)$.

3. For $i = 1$ to $i^* - 1$:

   (a) Send to $\mathcal{A}$ a sample $a_i = f(x, \tilde{y}_i)$, where $\tilde{y}_i \leftarrow \mathcal{Y}$ are all independent.[11]

   (b) If $\mathcal{A}$ aborts B, then send to the trusted party T a random input $x^* \leftarrow \mathcal{X}$ chosen uniformly at random, and receive w_i. Output $a_1, \ldots, a_i$ and $b_{i-1} := w_i$, and halt.

4. If $\mathcal{A}$ did not abort A, send x to T and obtain the output $w = f(x, y)$.

5. Send $a_{i^*} = w$ to $\mathcal{A}$. If it aborts, output $a_1, \ldots, a_i$ and $b_{i^*-1} = 1$ with probability β.

6. Otherwise, for $i = i^* + 1$ to r:

   (a) Send to $\mathcal{A}$ the output $a_i = w$.

   (b) If $\mathcal{A}$ aborts, output $a_1, \ldots, a_i$ and $b_{i-1} := w$, and halt.

7. If no abort occurred, output $a_1, \ldots, a_r$ and halt.

Clearly, the view generated in the ideal world until an abort occurred is identically distributed to the real world view. Since no honest party has an output, the simulator successfully simulates $\mathcal{A}$. We now consider the case where B and C are corrupted. The simulator does the following.

1. Query the adversary for the input y it sends to the dealer.

2. Sample $i^* \leftarrow \mathsf{Geom}(\alpha)$.

3. For $i = 1$ to $i^* - 1$:

   (a) Send to $\mathcal{A}$ a sample $b_i = f(\tilde{x}_i, y)$, where $\tilde{x}_i \leftarrow \mathcal{X}$ are all independent.[12]

   (b) If $\mathcal{A}$ aborts B, then send $y^* \leftarrow \mathcal{Y}$ to the trusted party T and receive w_i. Output $b_1, \ldots, b_i$ and $a_i := w_i$, and halt.

4. If $\mathcal{A}$ did not abort B, send y to T and obtain the output $w = f(x, y)$.

5. For $i = i^*$ to r:

   (a) Send to $\mathcal{A}$ the output $b_i = w$.

   (b) If $\mathcal{A}$ aborts, output $b_1, \ldots, b_i$ and $a_i := w$, and halt.

6. If no abort occurred, output $b_1, \ldots, b_r$ and halt.

Similarly to the previous case, the view generated in the ideal world until an abort occurred is identically distributed to the real world view. Therefore, the simulator successfully simulates the adversary.

[11] Formally, the simulator should send two shares of b_i in a 2-out-of-2 secret sharing scheme.
[12] Formally, the simulator should send two shares of b_i in a 2-out-of-2 secret sharing scheme.

B.2 Proof of Proposition 6.7

Proof of Proposition 6.7. We compute the difference $s(k+1)-s(k)$ and express it in simpler terms. It holds that

$$\begin{aligned}
s(k+1)-s(k) &= (-1)^{\frac{n}{2}+k+1}\cdot\binom{n}{n/2}\binom{\frac{n}{2}-1}{k} \\
&\quad+\sum_{m=k+1}^{\frac{n}{2}-1}(-1)^{m+k+1}\cdot\binom{n}{m}\cdot\left[\binom{m-1}{k}+\binom{n-m-1}{k}\right] \\
&\quad-(-1)^{\frac{n}{2}+k}\cdot\binom{n}{n/2}\binom{\frac{n}{2}-1}{k-1} \\
&\quad-\sum_{m=k}^{\frac{n}{2}-1}(-1)^{m+k}\cdot\binom{n}{m}\cdot\left[\binom{m-1}{k-1}+\binom{n-m-1}{k-1}\right] \\
&= (-1)^{\frac{n}{2}+k+1}\cdot\binom{n}{n/2}\binom{n/2}{k}-\binom{n}{k}\cdot\left[1+\binom{n-k-1}{k-1}\right] \\
&\quad+\sum_{m=k+1}^{\frac{n}{2}-1}(-1)^{m+k+1}\cdot\binom{n}{m}\cdot\left[\binom{m}{k}+\binom{n-m}{k}\right] \\
&= (-1)^{\frac{n}{2}+k+1}\cdot\binom{n}{k}\binom{n-k}{\frac{n}{2}-k}-\binom{n}{k}\cdot\left[1+\binom{n-k-1}{k-1}\right] \\
&\quad+\sum_{m=k+1}^{\frac{n}{2}-1}(-1)^{m+k+1}\cdot\binom{n}{k}\cdot\left[\binom{n-k}{m-k}+\binom{n-k}{m}\right] \\
&= \binom{n}{k}\cdot\Bigg[(-1)^{\frac{n}{2}+k+1}\cdot\binom{n-k}{n/2}-1-\binom{n-k-1}{k-1} \\
&\qquad+\sum_{m=k+1}^{\frac{n}{2}-1}(-1)^{m+k+1}\cdot\left[\binom{n-k}{m-k}+\binom{n-k}{m}\right]\Bigg], \qquad (5)
\end{aligned}$$

where the second equality is by Pascal's identity, and the third and the fourth follow from Proposition 6.4. Now, continuing the summation of the first term we obtain

$$\sum_{m=k+1}^{\frac{n}{2}-1}(-1)^{m+k+1}\cdot\binom{n-k}{m-k}=\sum_{m=1}^{\frac{n}{2}-k-1}(-1)^{m+1}\cdot\binom{n-k}{m}=-(-1)^{\frac{n}{2}-k-1}\cdot\binom{n-k-1}{\frac{n}{2}-k-1}+1,$$

where last equality is by Proposition 6.5. Similarly, by Proposition 6.5 the summation of the second term second equals to

$$\begin{aligned}
\sum_{m=k+1}^{\frac{n}{2}-1}(-1)^{m+k+1}\cdot\binom{n-k}{m} &= (-1)^{k+1}\cdot\left[(-1)^{\frac{n}{2}-}\cdot\binom{n-k-1}{\frac{n}{2}-1}-(-1)^{k}\binom{n-k-1}{k}\right] \\
&= (-1)^{\frac{n}{2}+k}\cdot\binom{n-k-1}{\frac{n}{2}-1}+\binom{n-k-1}{k}.
\end{aligned}$$

Plugging it back into Equation (5) we obtain

$$\begin{aligned}
s(k+1)-s(k) &= \binom{n}{k}\cdot\Bigg[(-1)^{\frac{n}{2}+k+1}\cdot\binom{n-k}{n/2}-1-\binom{n-k-1}{k-1} \\
&\qquad + \sum_{m=k+1}^{\frac{n}{2}-1}(-1)^{m+k+1}\cdot\left[\binom{n-k}{m-k}+\binom{n-k}{m}\right]\Bigg] \\
&= \binom{n}{k}\cdot\Bigg[(-1)^{\frac{n}{2}+k+1}\cdot\binom{n-k}{n/2}-1-\binom{n-k-1}{k-1} \\
&\qquad -(-1)^{\frac{n}{2}-k-1}\cdot\binom{n-k-1}{\frac{n}{2}-k-1}+1 \\
&\qquad +(-1)^{\frac{n}{2}+k}\cdot\binom{n-k-1}{\frac{n}{2}-1}+\binom{n-k-1}{k}\Bigg] \\
&= \binom{n}{k}\cdot\Bigg[(-1)^{\frac{n}{2}+k+1}\cdot\binom{n-k}{n/2}+\binom{n-k-1}{k}-\binom{n-k-1}{k-1} \\
&\qquad +(-1)^{\frac{n}{2}-k}\cdot\binom{n-k-1}{n/2}+(-1)^{\frac{n}{2}+k}\cdot\binom{n-k-1}{\frac{n}{2}-1}\Bigg] \\
&= \binom{n}{k}\cdot\Bigg[(-1)^{\frac{n}{2}+k+1}\cdot\binom{n-k}{n/2}+\binom{n-k-1}{k}-\binom{n-k-1}{k-1} \\
&\qquad +(-1)^{\frac{n}{2}+k}\cdot\binom{n-k}{n/2}\Bigg] \\
&= \binom{n}{k}\binom{n-k-1}{k-1}\cdot\frac{n-3k}{k},
\end{aligned}$$

where the fourth is by Pascal's identity. Clearly, the above expression is non-negative if and only if $k \le n/3$. □