On Fully Secure MPC with Solitary Output

Shai Halevi[*] Yuval Ishai[†] Eyal Kushilevitz[‡] Nikolaos Makriyannis[§]
Tal Rabin[¶]

September 19, 2019

Abstract

We study the possibility of achieving *full security*, with guaranteed output delivery, for secure multiparty computation of functionalities where only one party receives output, to which we refer as *solitary functionalities*. In the standard setting where all parties receive an output, full security typically requires an honest majority; otherwise even just achieving fairness is impossible. However, for solitary functionalities, fairness is clearly not an issue. This raises the following question: Is full security with no honest majority possible for *all* solitary functionalities?

We give a negative answer to this question, by showing the existence of solitary functionalities that cannot be computed with full security. While such a result cannot be proved using fairness-based arguments, our proof builds on the classical proof technique of Cleve (STOC 1986) for ruling out fair coin-tossing and extends it in a nontrivial way.

On the positive side, we show that full security against any number of malicious parties is achievable for many natural and useful solitary functionalities, including ones for which the multi-output version cannot be realized with full security.

[*] Algorand Foundation. Work done while at IBM Research.

[†] Department of Computer Science, Technion. E-mail: yuvali@cs.technion.ac.il.

[‡] Department of Computer Science, Technion. E-mail: eyalk@cs.technion.ac.il.

[§] Department of Computer Science, Technion. E-mail: n.makriyannis@gmail.com.

[¶] Algorand Foundation. Work done while at IBM Research.

Contents

1 Introduction

Secure multiparty computation (MPC) [31, 19, 7, 9] allows a set of mutually distrusting parties to compute any function of their local inputs while guaranteeing (to the extent possible) the privacy of the the inputs and the correctness of the outputs. Security is formulated by requiring that a real execution of a protocol is indistinguishable from an ideal execution in which the parties hand their inputs to a trusted party who computes the function and returns the outputs.

The strongest level of security one could hope for is so-called "full security" [19, 8]. Full security ensures *guaranteed output delivery* in the sense of allowing all parties to learn their outputs without revealing additional information about other inputs. In particular, it implies *fairness*: malicious parties cannot learn their outputs while preventing honest parties from learning their outputs. This level of security is achievable in the presence of an honest majority, either unconditionally [7, 9, 4, 30] (assuming secure point-to-point channels and a broadcast channel) or under standard cryptographic assumptions [19, 18] (assuming a public-key infrastructure).

Without an honest majority, a classical result of Cleve [11] shows that full security, or *even fairness alone*, is generally impossible. Concretely, there are many natural functionalities such that in *every protocol* for computing them, malicious parties can gain a significant advantage over honest parties in learning information about the output. Thus, when no honest majority is assumed, it is common to settle for weaker notions of security such as "security with abort" [31, 19, 5, 20, 21].

In this paper, we consider the possibility of achieving full security for functionalities that deliver output to a single party, to which we refer as "functionalities with solitary output" or "solitary functionalities" for short. Such functionalities capture many realistic use-cases of MPC in which different participants play different roles. For instance, consider a (single) employer who wishes to learn some aggregate private information about a group of employees, where the output should remain hidden from the employees. This type of functionalities is commonly considered in the non-interactive setting, including the Private Simultaneous Messages (PSM) model of secure computation [15] and its robust variants [6, 1].

Beyond being a natural class of functionalities, the class of solitary functionalities is also interesting because it bypasses all fairness-based impossibility results. Indeed, fairness is not an issue when only one party receives an output, and thus Cleve's impossibility result does not have any consequences for such functionalities. Therefore, the first question that we ask is a very basic feasibility question in the theory of MPC:

> Do *all* functionalities with solitary output admit a fully secure protocol?

This feasibility question can be contrasted with the state of affairs in other ongoing lines of work on characterizing the functionalities that admit protocols with *information-theoretic* security, or *UC security*, or *fairness* [27, 10, 23, 3, 13], where the high-order bit is already known and the current efforts are focused on trying to fully characterize the realizable functionalities.

We make two main contributions. On the negative side, we settle the high-order bit by proving that some solitary functionalities *cannot* be computed with full security. This is conceptually intriguing because, as mentioned above, solitary functionalities do not introduce "fairness" problems. So what is the source of difficulty in achieving full security? Our impossibility proof extends Cleve's original attack in a rather subtle way. In Cleve's attack, the adversary gains advantage over honest parties by aborting the protocol at a point where it knows significantly more information about the output than the honest parties do. Our new attack, dubbed the "double-dipping attack", is based on the following rough intuition. (The following simplified description of the attack ignores important subtleties; see Section 1.2 and Section 3 for a more precise version.) The adversary controls a majority of the parties that includes the output party. It instructs one of the parties it controls to abort the protocol just when learning enough (but not all) information about the output. Intuitively, in such a case, the protocol must be run again with default values (in particular, the original inputs cannot be recovered as the aborting parties form a majority). In the end of the protocol, the adversary learns the output of f on *two* inputs, with the same input values for honest parties. This is an information that the adversary cannot obtain in the ideal world, hence security fails.

On the positive side, we make progress towards full characterization of the solitary functionalities that admit fully secure protocols. We present such protocols for several natural and useful families of solitary functionalities, including variants of commonly studied MPC problems such as Private Set Intersection. Our positive results apply in many cases where negative results are known for the multi-output variant. We elaborate on both our positive and negative results below.

1.1 Our Results

For our negative result, we present a family Ω of solitary functionalities for which no fully secure protocol exists. A representative example of such a functionality, first considered in the context of "best of both worlds" security [24] (see below), is the following 3-party functionality f_{eq} with two parties P_1 and P_2 receiving inputs $x, y \in \{1, 2, 3\}$, respectively, and an output-receiving party Q. The output of f_{eq} is defined as $f_{\mathsf{eq}}(x, y) = x$ if $x = y$ and $f_{\mathsf{eq}}(x, y) = \perp$ otherwise. We sketch below how "double dipping" is applied to this functionality, and present the family Ω and the formal impossibility proof in Section 3.

Next, in Section 4, we present several positive results. We start by proving that fairness implies full security in the following sense: if f is an n-party function, where all parties receive the output, and f can be computed with fairness, then the $(n+1)$-party solitary functionality f', with inputs given to $P_1, \ldots, P_n$, as in f, and with the output delivered to the output party Q, can be computed with full security. Our next positive result shows that we can go much beyond fairness positive results; specifically, we consider a family of n-party functionalities that we call functions with "forced output distribution". Described for the 3-party case, this family includes all functions $f(x, y)$ (with inputs x, y to P_1, P_2, respectively, and output to Q) such that for at least one of the input parties, say P_1, there is a distribution on its input, where the output $f(x, y)$ is distributed the same, no matter what the other input is. Note that such (non-trivial) functions f cannot be computed with fairness, as this would imply fair coin-tossing, which is impossible [11]. Finally, as a third positive result, we consider a family of functionalities that we term "functionalities with fully revealing input". Described in the 3-party setting above, this family includes all functionalities where one of the parties, say P_1, has an input for which the function f becomes injective.

We stress that these results fall short of providing a full characterization of the fully secure solitary functionalities, as we give an example of a function that does not fall into any of the families of positive results but nevertheless can be computed with full security. Interestingly, we compute this function using a variant of the GHKL protocol [23] for computing fair two-party functionalities, yet — viewed as a symmetric two-party functionality — it is inherently unfair. We leave the question of finding a full characterization as an intriguing open question for future work.

Example: To demonstrate the usefulness of the above positive and negative results, we consider some variants of the Private Set Intersection (PSI) problem. In this problem, the inputs x, y of P_1, P_2

correspond to subsets S_1, S_2 of some domain $[m]$ and the output is the intersection $S = S_1 \cap S_2$. It follows from our negative result that if $|S_1| = |S_2| = k$, for some fixed k, then this function cannot be computed with full security (in fact, the function f_{eq} mentioned above is exactly the case $k = 1$). On the other hand, for the same inputs, if the required output is only the intersection size, i.e. $|S|$, then this becomes a functionality with a forced output distribution (e.g., by choosing S_1 as a uniformly random set of size k) and so this functionality can be computed with full security. Similarly, if we allow $|S_1|, |S_2|$ to be anywhere between k and m then PSI with full security becomes possible (using $[m]$ as a revealing input) and, if we allow $|S_1|, |S_2|$ to be anywhere between 0 and k, this is also possible (using a degenerate version of the forced output distribution, where $\emptyset$ is selected with probability 1). Other interesting cases, like the case where $|S_1|, |S_2|$ are between 1 and k, are left as an open problem. (See Appendix 5, for an analysis of additional variants of PSI, including additional variants where the output is just the intersection size $|S|$, or just a bit indicating whether $S = \emptyset$, sometimes referred to as the *disjointness* function. The Appendix also includes similar analyses for various useful flavors of Oblivious Transfer (OT).)

Classification of PSI variants

Define $\mathsf{PSI}^g_m : (S_1, S_2) \mapsto g(S_1 \cap S_2)$, for some $m \in \mathbb{N}$ and function g, where S_1, $S_2 \subseteq [m]$ are drawn from a predetermined input space (specified below). Let $k < \ell$ denote some arbitrary fixed numbers different than 0 or m such that $k + \ell + 1 \leq m$.

Input restriction. \ Function g	$S \mapsto S$	$S \mapsto \lvert S \rvert$	$S \mapsto \begin{cases} 1 & \text{if } S \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$
$\lvert S_1 \rvert, \lvert S_2 \rvert = k < m/2$	Impossible	Forced	Forced
$\lvert S_1 \rvert, \lvert S_2 \rvert \in \{0, \ldots, k\}$	Forced	Forced	Forced & Fairness
$\lvert S_1 \rvert, \lvert S_2 \rvert \in \{k, \ldots, m\}$	Revealing	Open	Forced & Fairness
$\lvert S_1 \rvert, \lvert S_2 \rvert \in \{k, \ldots, \ell\}$	Open	Open	Open
$\lvert S_1 \rvert \in \{k, \ldots, \ell\}$ and $\lvert S_2 \rvert \in \{k, \ldots, \ell + 1\}$	Open	Open	Fairness

Figure 1: Table Summarizing our Results vis-à-vis the PSI Problem

Finally, as an additional contribution, we analyse the round complexity of computing solitary functionalities with full security. We observe that some of the protocols presented in our positive results are constant-round protocols, while others use super-logarithmic number of rounds. We prove that, for certain solitary functionalities, full security actually requires super-constant round complexity (see Section 5). We leave the question of figuring out the exact round-complexity for any solitary functionality as an intriguing open question for future work.

Feasibility landscape of Boolean solitary functionalities. We conclude this section with a few sentences regarding the "feasibility" landscape of solitary MPC. We focus on functions with Boolean output where the output receiving party does not provide input; this case is interesting as it is readily comparable to the non-solitary Boolean two-party case (the most well understood instance of fully secure MPC with dishonest majority). We distinguish two cases depending of the size of the input domains. From the fairness criterion, if one party has a strictly bigger input domain than the other, then almost all functionalities are computable with full security, because almost all two-party Boolean functions admit fair protocols in this case [3]. On the other hand, when the parties have exactly the same number of inputs, the fairness criterion does *not* apply, because almost all two-party Boolean functions are *not* computable with fairness.[1] However, by excluding the functions that are computable using a variant of the forced criterion, we can succinctly describe the set of functions whose status is unknown: $\{M \in \{0,1\}^{n\times n} \mid \exists \mathbf{x} \in \mathbb{R}^n \text{ s.t. } M\mathbf{x} = \mathbf{1}_n \wedge \sum_i \mathbf{x}_i \leq 0\}$. In words, the set corresponds to 0-1 matrices (viewed as matrices over the reals) whose columns span $\mathbf{1}_n$ with coefficient that have a negative sum. While we could not rigorously analyze the measure of this set, we conjecture that it represents a vanishing fraction of the entire space, i.e. relative to $\{0,1\}^{n\times n}$; experimental evidence for $n \leq 300$ strongly supports our conjecture (see Appendix 5). Thus, the following picture emerges for functionalities with equal-sized input domains: almost all 2-party functionalities *cannot* be computed fairly, while almost all solitary 3-party functionalities (two inputs and one output) *can* be computed with full security.

1.2 Our Techniques.

Next, we elaborate on some of the techniques that we use.

(i) Impossibility result. As mentioned above, for our impossibility result, we use a technique inspired by Cleve's seminal "biasing" attack on coin-tossing [11]. In Cleve's attack, the adversary is trying to bias the output of a fair coin-flip. The adversary picks a random round i, and plays honestly until that round. Then, the adversary computes the corrupted party's backup value for that round, i.e. the output prescribed by the protocol in case the other party aborted at that round. The adversary aborts the corrupted party at that round or the next round depending on the "direction" it is attempting to bias the output to. Intuitively, because the protocol is inherently unfair, the adversary has an advantage in learning the output. Therefore, by aborting prematurely, the adversary alters the distribution of the honest party's output.

Translating the above attack to our setting is not straightforward, given that the above gives an attack on correctness while we aim for an attack on privacy. For concreteness, we now explain how our impossibility applies to the 3-party functionality f_{eq} described above. Notice that, in an ideal execution, if P_1 chooses its input at random, then the other two colluding parties can only be sure of P_1's input with probability at most 1/3 (i.e. by guessing the right value). In the real-world however, there must be some round of the protocol where the joint backup value of P_2 and Q (i.e. the output prescribed by the protocol in case P_1 aborted at that round) contains information about P_1's input, while the joint backup value of P_1 and Q does not contain information about P_2's input. By aborting P_2 at that round, the adversary can effectively compute the output on two different inputs of P_2 and thus guess P_1's input with probability noticeably greater than 1/3.

Rather crudely, the above can be summarized as follows: We define a coin-toss between $\{P_1, Q\}$ and $\{P_2, Q\}$ such that the outcome of the "coin-toss" is tied to some privacy event. By "biasing" the coin-toss, the adversary effectively increases its chance that the privacy event occurs, which results in a privacy breach. It should be noted that this picture is not accurate since, in our setting,

[1] The reason being that most such functions can be used to implement the coin-tossing functionality [28] – which does not admit a fair protocol.

the direction of bias is very important and this cannot be guaranteed by Cleve's attack.

(ii) Protocols. Our transformation from n-party fair protocols (with output to all) to $(n+1)$-party fully secure protocols with solitary output to Q describes a compiler that takes a fair protocol Π and transforms it into a fully secure protocol Π' with solitary output. The idea is to emulate Π by sharing the view of each party P_i in the original protocol Π between P_i and Q in Π'. This way, an adversary corrupting a subset of parties not including Q learns nothing, while an adversary corrupting a subset of parties that includes Q only learns the views of the corresponding parties in Π. The latter cannot be used to mount an attack, given the presumed security of the original protocol. Our protocols for the forced output distribution class and for the fully revealing input class are very different. Interestingly, these two cases are symmetric in some sense, where each has "problematic" parties. In the former (forced output distribution) case, the problematic party is the one that does not have a forced output distribution. The protocol we propose in this case funnels the communication through the others parties. Thus, by design, the problematic party only contributes to the computation once. For the latter (fully revealing input) case, the problematic parties are the ones without fully revealing input. The protocol we propose for this case funnels the communication through the party with a revealing input, say P_1. Thus, by design, unless P_1 is corrupt (in which case there are no secrets), computation only occurs once.

Related Work. Below, we discuss some related work that deals with full security and other related security notions (in particular, fairness).

In the two-party case, it is known that fairness is equivalent to full security (with guaranteed output delivery), since if an honest party aborts it can safely replace the input of the corrupted party by a default value and compute the resulting output locally. In contrast, Cohen and Lindell [12] show that in the multiparty case there are functionalities that admit fair protocols but do not admit fully secure protocols.

Since the work of Cleve [11], it is known that full security, or even fairness, cannot be achieved in general unless there is an honest majority. This led to a rich line of work [23, 29, 2, 3, 14] attempting to characterize *which* functions can be computed with full security. Most works along this line focused on the two-party case, starting with the results of [23], and culminating in a full characterization for the class of fair Boolean functions with the same output for both parties [3].

Less is known for the multi-party case. Examples of multi-output functions for which fair protocols exist (specifically, n-party OR and 3-party majority) are given in [22]. In [24, 26] (see also [25]), the notion of "Best-of-both-worlds security" is introduced as a hybrid between full security and security with abort. A protocol satisfies this definition, if there is one protocol that *simultaneously* provides full security if there is an honest majority and otherwise it guarantees security with abort. Note that, in the context of best-of-both-worlds, [24] already gives an example of a 3-party solitary function for which no constant-round protocol exists (concretely, the function f_{eq} mentioned above). This was improved to $\log n$ rounds in [26].

Open Problems. As mentioned above, the most obvious open problems are obtaining a characterization or at least reducing the gap between the positive and negative results, and working out the exact round complexity for fully secure computation of solitary functionalities. Less obviously, we identify the following interesting open questions.

1. Our attack in Section 3 crucially relies on the *rushing* capability of the adversary. It would

be interesting to show that this is inherent for impossibility or to extend the negative result to the case of a non-rushing adversary.

2. In this work, we are mainly concerned with the feasibility questions of solitary MPC. Therefore, for obtaining malicious security, our protocols use a generic step that we have not tried to optimize. We leave the interesting question of improving concrete efficiency for future work, or designing concretely efficient fully secure protocols for useful special cases such as PSI.

3. As explained in subsequent sections, broadcast is necessary for solitary MPC. However, *some* functionalities do not require broadcast. While the question is orthogonal to the goal of the paper, it would be interesting to understand which functionalities require broadcast in the solitary setting.

2 Preliminaries

The following models and definitions are adapted from [17, 12].

2.1 Models

In this section we outline the definition of secure computation, following Canetti's definition approach for the standalone model [8], and highlight some details that are important for our purposes. The following version of the definition is somewhat simplified. We refer the reader to [8] for more complete definitions.

Communication model. We consider a network of n processors, usually denoted $P_1, \ldots, P_n$ and referred to as *parties*. Each pair of parties is connected via a private, authenticated point-to-point channel. In addition, all parties share a common *broadcast* channel, which allows each party to send an identical message to *all* other parties. In some sense, the broadcast channel can be viewed as a medium which "commits" the party to a specific value.[2]

Functionality. A secure computation task is defined by some *n-party functionality* $f : X_1 \times \ldots \times X_n \to \Sigma^n$, specifying the desired mapping from the parties' inputs to their final outputs. Party P_i's input domain is denoted by X_i, for each $i \in [n]$, and the outputs of the parties are assumed to belong to some alphabet Σ. When $n = 3$, the parties' input domains will be denoted X, Y and Z to make the distinction more explicit. One may also consider *randomized* functionalities, which take an additional random input; however, in this work we focus on the deterministic case.

Functionality with Solitary Output. A *n-party functionality* $f : X_1 \times \ldots \times X_n \to \Sigma^n$ admits solitary output if it delivers output to (the same) one party alone, i.e. f is of the form $(x_1, \ldots, x_n) \mapsto (\emptyset, \ldots, \emptyset, \sigma, \emptyset, \ldots, \emptyset)$, where the index of σ does not depend on the input. The output-receiving party will be denoted by, Q, and, unless stated otherwise, will be identified with P_n. If no confusion arises, we simply write $f : X_1 \times \ldots \times X_n \to \Sigma$ or $f : (x_1, \ldots, x_n) \mapsto \sigma$.

[2] We remark that our assumption regarding broadcast is in fact necessary for fully secure computation of solitary functionalities. This observation follows from the fact that "convergecast" implies broadcast [16].

Some Notations. Denote by $\mathcal{P} = \{P_1, \ldots, P_n\}$ the set of all parties. If no confusion arises, we sometimes identify $\mathcal{P}$ with the numbers in $[n] = \{1, \ldots, n\}$. Subsets of these parties are denoted by calligraphic letters $(\mathcal{S}, \mathcal{T}, \ldots)$, and their complements will be denoted by $(\overline{\mathcal{S}}, \overline{\mathcal{T}}, \ldots)$. Random variables are denoted by lower-case boldface $(\mathbf{x}, \mathbf{y}, \ldots)$ and distributions by upper-case boldface $(\mathbf{X}, \mathbf{Y}, \ldots)$. For a functionality f taking input from $X_1 \times \ldots \times X_n$ we will write $x_{\mathcal{S}}$ to denote an element of the subspace $\times_{i \in \mathcal{S}} X_i$ and, abusing notation, $f(x_{\mathcal{S}}, x_{\overline{\mathcal{S}}})$ denotes the value of $f(x_1, x_2, \ldots, x_n)$. Furthermore, for integers m and k, we let $\binom{[m]}{k}$ denote the subsets of $[m]$ of size exactly k and $2^{[m]}$ the set of all subsets of $[m]$. For set $\mathcal{S}$ and distribution $\mathbf{S}$, we write $s \leftarrow \mathcal{S}$ and $s \leftarrow \mathbf{S}$ to denote that element s is sampled uniformly at random from $\mathcal{S}$ or according to distribution $\mathbf{S}$, respectively.

Protocol. Initially, each party P_i holds an input x_i, a random input ρ_i and, possibly, a common *security parameter* κ. The parties are restricted to (expected) polynomial time in κ. The protocol proceeds in *rounds*, where in each round each party P_i may send a "private" message to each party P_j (including itself) and may broadcast a "public" message, to be received by all parties. The messages P_i sends in each round may depend on all its inputs (x_i, ρ_i and κ) and the messages it received in previous rounds. Without loss of generality, we assume that each P_i sends x_i, ρ_i, κ to itself in the first round, so that the messages it sends in each subsequent round may be *determined* from the messages received in previous rounds. We assume that the protocol terminates after a fixed number of rounds, denoted r (that may depend on the security parameter κ), and that honest parties never halt prematurely, i.e. honest parties are active at any given round of the protocol. Finally, each party locally computes some output based on its view. We note that our negative results extend to protocols that have *expected polynomial* number of rounds (in κ) via a simple Markov inequality argument.

Fail-Stop Adversary. We consider a *fail-stop t-adversary $\mathcal{A}$*, where the parameter t is referred to as the *security threshold*. The adversary is an efficient interactive algorithm,[3] which is initially given the security parameter κ and a random input ρ. Based on these, it may choose a set $\mathcal{T}$ of *at most t* parties to corrupt. The adversary then starts interacting with a protocol (either a "real" protocol as above, or an *ideal-process* or *hybrid-process* protocol to be defined below), where it takes control of all parties in $\mathcal{T}$. In particular, it can read their inputs, random inputs, and received messages and, contrary to the *malicious* case (see below), it can control the messages that parties in $\mathcal{T}$ send only by deciding whether to send them or to abort. We assume by default that the adversary has a *rushing* capability: at any round it can first wait to hear all messages sent by uncorrupted parties to parties in $\mathcal{T}$, and use these to make its decisions whether to abort or continue (some of) the parties he corrupts. Corrupted parties that do not abort send their prescribed messages for the present round, while corrupted parties that abort send a special abort symbol to *all parties*.[4]

Malicious Adversaries. Adversaries that deviate arbitrarily from the protocol are not discussed in the present paper. Using the GMW compiler [19], our positive results can be extended to

[3] It is usually assumed that the adversary is given an "advice" string a, or is alternatively modeled by a nonuniform algorithm. In fact, the proofs of our negative results are formulated in this nonuniform setting, but can be modified to apply to the uniform one as well.

[4] This assumption implies that an abort is detected by all parties, even one that occurred on a private channel. This assumption can be enforced via a dispute resolution mechanism, thanks to the broadcast channel.

malicious adversaries. Negative results trivially extend to such adversaries (since fail-stop is a special kind of malicious adversary).

Security. We consider two types of security known as *full security* and *security with identifiable abort*. The former is the focus of the paper, i.e. it corresponds to the security notion we want to realize or rule out. The latter is a weaker security notion that is useful towards realizing our positive results. Informally, a protocol computing f is said to be t-secure if whatever a t-adversary can "achieve" by attacking the protocol, it could have also achieved (by corrupting the same set of parties) in an ideal process in which f is evaluated using a trusted party. To formalize this definition, we have to define what "achieve" means and what the ideal process is. The *ideal process* for evaluating the functionality f is a protocol π_f involving the n parties and an additional, incorruptible, trusted party TP.

Ideal model with full security. The protocol proceeds as follows: (1) each party P_i sends its input x_i to TP; (2) TP computes f on the inputs (using its own random input in the randomized case), and sends to each party its corresponding output. Note that when the adversary corrupts parties $\mathcal{T}$ in the ideal process, it can pick the inputs sent by parties in $\mathcal{T}$ to TP (possibly, based on their original inputs) and then output an arbitrary function of its view (including the outputs it received from TP). Honest parties always output the message received from the trusted party and the corrupted parties output nothing.

Ideal model with identifiable abort. In this case, an adversary can abort the computation in the ideal model after learning its outputs, at the cost of revealing to the honest parties the identity of at least one of the corrupted parties. The protocol proceeds as follows: (1) each P_i sends its input x_i to TP; (2) TP computes f on the inputs (using its own random input in the randomized case), and sends to each of the *corrupted* parties its corresponding output. (3) By sending to TP either $(\mathsf{continue}, \emptyset)$ or (abort, P_i), for some P_i in $\mathcal{T}$, according to whether the adversary continues the execution, or aborts the execution at the cost of revealing one corrupted party. (4) TP sends the outputs to the honest parties if the adversary continues, or the identity of the corrupted P_i together with a special abort-symbol, if the adversary aborted the computation. Similarly to the previous case, when an adversary corrupts parties in the ideal process, it can pick the inputs sent by parties in $\mathcal{T}$ to TP (possibly, based on their original inputs) and then output an arbitrary function of its view (including the outputs it received from TP). Honest parties always output the message received from the trusted party and the corrupted parties output nothing.

2.2 Security Definition

To formally define security, we capture what the adversary "achieves" by a random variable concatenating the adversary's output together with the *outputs* and the *identities* of the uncorrupted parties. For a protocol Π, adversary $\mathcal{A}$, input vector x, and security parameter κ, let $exec_{\Pi,A}(\kappa, x)$ denote the above random variable, where the randomness is over the random inputs of the uncorrupted parties, the trusted party (if f is randomized), and the adversary. The security of a protocol Π (also referred to as a *real-life* protocol) is defined by comparing the *exec* variable of the protocol Π to that of the ideal process π_f^{type}, where $\mathsf{type} \in \{\mathsf{full\_sec}, \mathsf{id\_abort}\}$ specifies the ideal process to be compared with (either full security or identifiable abort). Formally:

Definition 2.1. *We say that a protocol* Π t-securely computes f *if, for any (real-life) t-adversary $\mathcal{A}$, there exists (an ideal-process) t-adversary $\mathcal{A}'$ such that the distribution ensembles* $exec_{\Pi,\mathcal{A}}(\kappa, x)$ *and* $exec_{\pi_f^{\mathsf{type}},\mathcal{A}'}(\kappa, x)$ *are* indistinguishable. *The security is referred to as perfect, statistical, or computational according to the notion of indistinguishability being achieved. For instance, in the computational case it is required that for any family of polynomial-size circuits $\{C_\kappa\}$ there exists some* negligible *functionality* neg*, such that for any x,*

$$|C_\kappa(exec_{\Pi,\mathcal{A}}(\kappa, x)) - C_\kappa(exec_{\pi_f^{\mathsf{type}},\mathcal{A}'}(\kappa, x))| \le \mathsf{neg}(\kappa).$$

An equivalent form of Definition 2.1 quantifies over all *input distributions* X rather than specific input vectors x. This equivalent form is convenient for proving our negative results.

Intuitive discussion. Definition 2.1 asserts that for any *real-life* t-adversary $\mathcal{A}$ attacking the real protocol there is an *ideal-process* t-adversary $\mathcal{A}'$ which can "achieve" in the ideal process as much as $\mathcal{A}$ does in the real life. The latter means that the output produced by $\mathcal{A}'$ together with the inputs and outputs of uncorrupted parties in the ideal process is indistinguishable from the output (wlog, the entire view) of $\mathcal{A}$ concatenated with the inputs and outputs of uncorrupted parties in the real protocol. This concatenation captures both *privacy* and *correctness* requirements. On the one hand, it guarantees that the view of $\mathcal{A}$ does not allow it to gain more information about inputs and outputs of uncorrupted parties than is possible in the ideal process and, on the other hand, it ensures that the inputs and outputs of the uncorrupted parties in the real protocol be consistent with some correct computation of f in the ideal process. We stress that ideal-world adversary can indeed choose whatever input it likes, and it need not restrict itself to the input chosen by the real-world adversary.

Default Security Threshold. Throughout the paper, we assume that the security threshold is $t = n - 1$, namely an arbitrary strict subset of the parties can be corrupted. We therefore do not mention the parameter t in the rest of the paper.

2.3 Hybrid Model & Composition

Hybrid Model. The hybrid model extends the real model with a trusted party that provides ideal computation for predetermined functionalities. In more detail, the parties communicate with this trusted party as per the specifications of the ideal models described above (either fully secure or identifiable abort, to be specified). Let Fn be a functionality. Then, an execution of a protocol Π computing a functionality f in the Fn-hybrid model involves the parties interacting as per the real model and, in addition, having access to a trusted party computing Fn. The protocol proceeds in rounds such that, at any given round, the parties send normal messages as in the standard model, or, make a single invocation of the functionality Fn. Security is defined analogously to Definition 2.1 by replacing the real protocol with the hybrid one. The model in question is referred to as the $(\mathsf{Fn}, \mathsf{type})$-hybrid model, depending on the specification of the ideal functionality.

Composition. The hybrid model is useful because it allows cryptographic tasks to be divided into subtasks. In particular, a fully secure hybrid protocol making ideal invocations to an ideal functionality with identifiable abort can be transformed into a fully secure real protocol, if there exists a real protocol for the ideal functionality that is secure with identifiable abort. This technique is captured by Canneti's sequential composition theorem.

Theorem 2.2 (Canetti [8])**.** *Suppose that protocol* Π securely computes f *in the* $(\mathsf{Fn}, \mathsf{id\_abort})$*-hybrid model with full security, and suppose that* Ψ securely computes f *in the real model. Then, protocol* Π^{Ψ} securely computes f *in the real model, where* Π^{Ψ} *is obtained by replacing ideal invocations of* Fn *with real executions of* Ψ*. Furthermore, the quality of the security (computational, statistical or perfect) of the resulting protocol is the weakest among the security of* Π *and* Ψ*.*

Finally, we define the notion of *backup values*. It is immediate from the security definition that any fully secure protocol admits well defined backup values.

Definition 2.3 (Backup values)**.** *The following definitions are with respect to a fixed honest execution of an* n*-party,* r*-round correct protocol (determined by the parties' random coins) for solitary functionality* f*. The* i^{th} $\mathsf{round\ backup\ value}$ *of a subset of parties* $\mathcal{Q} = \{Q\} \cup \mathcal{S} \subseteq \mathcal{P}$ *at round* $i \in [r]$*, denoted* $\mathsf{Backup}(\mathcal{Q}, i)$*, is defined as the value* Q *would output, if all parties in* $\mathcal{P} \setminus \mathcal{Q}$ *abort at round* $i+1$ *and no other party aborts. For consistency, we let* $\mathsf{Backup}(\mathcal{Q}, r)$ *denote the output of the protocol if no parties abort (i.e* $\mathsf{Backup}(\mathcal{Q}, r) = \mathsf{Backup}(\mathcal{Q}', r)$*, for every* $\mathcal{Q}$ *and* $\mathcal{Q}'$*).*

3 Impossibility: The Double-Dipping Attack

In this section we prove our main negative result. Namely, we show impossibility of achieving full security for a number of solitary functionalities, including the following natural families:

- Equality testing with leakage of input (including f_{eq} from the introduction).
- Private Set Intersection for fixed input size (i.e. PSI as defined in Definition 3.1).

Definition 3.1. *Let* $\mathsf{PSI}^{\text{id}}_{m,k} : \binom{[m]}{k} \times \binom{[m]}{k} \to 2^{[m]}$ *be such that* $\mathsf{PSI}^{\text{id}}_{m,k}(S_1, S_2) = S_1 \cap S_2$*. As a three party functionality,* $\mathsf{PSI}^{\text{id}}_{m,k}$ *receives inputs from* P_1 *and* P_2 *and delivers output to an additional party* Q*.*

Namely, $\mathsf{PSI}^{\text{id}}_{m,k}$ takes as input two sets of size k and outputs their intersection. We point out that $f_{\mathsf{eq}} \equiv \mathsf{PSI}^{\text{id}}_{m,1}$. In this section, we show impossibility for a class of functions that includes $\mathsf{PSI}^{\text{id}}_{m,k}$, for every $0 < k < m/2$. As a warm-up, we sketch our impossibility result for the specific functionality f_{eq}; the general case is essentially an extrapolation of this case. We will be using the following notation.

Notation 3.2. *Let* Π *be a three-party,* r*-round protocol for computing a function* $f : X \times Y \times Z \to \Sigma$ *with solitary output. Define random variables* $\mathbf{a}_0, \ldots, \mathbf{a}_r$ *and* $\mathbf{b}_0, \ldots, \mathbf{b}_r$ *such that* $\mathbf{a}_i$ *is the value of* $\mathsf{Backup}(\{Q, P_1\}, i)$ *in a random execution of* Π *and, similarly,* $\mathbf{b}_i$ *is the value of* $\mathsf{Backup}(\{Q, P_2\}, i)$ *in a random execution of* Π*, where* $\mathsf{Backup}(\mathcal{Q}, i)$ *is according to Definition 2.3.*

3.1 Warm Up

Let Π be a three-party protocol for computing f_{eq}. Let $\mathbf{X}$ and $\mathbf{Y}$ denote the uniform distribution for the inputs of P_1 and P_2 respectively. We proceed under the following simplifying assumptions for Π: for every $i \in [r]$, it holds that $\Pr_{x \leftarrow \mathbf{X}}[\mathbf{a}_i = x] = 1/3$ and $\Pr_{y \leftarrow \mathbf{Y}}[\mathbf{b}_i = y] = 1/3$. In words, if

P_1 (resp. P_2) chooses its input uniformly at random, then the backup output of Q and P_1 (resp. Q and P_2) at round i is equal to the aforementioned input with probability exactly 1/3, regardless of P_2's (resp. P_1's) choice of input. For the purposes of the present warm up, we will further assume that $\mathbf{a}_0$ and $\mathbf{b}_0$ are independent random variables. Next, we rule out fully secure computation for f_{eq} under these simplifying assumptions. When we tackle the general case in the next subsection, we get rid of these simplifying assumptions, by showing additional attacks (adversaries) where the aforementioned properties do no to hold.

We show that there exists an adversary that can guess the honest party's input with probability noticeably greater than what the ideal model allows. First, in the ideal model with full security, notice that when an honest party P_ℓ chooses his input uniformly at random, then an adversary corrupting $\{P_{3-\ell}, Q\}$ may guess (with certainty) the honest party's input with probability at most 1/3 (by using the right input for the corrupted party). We show that for any real protocol, there exists an adversary that can guess the input with noticeably greater probability, thus violating security.

Consider two adversaries A^{P_1} and A^{P_2} corrupting $\{Q, P_1\}$ and $\{Q, P_2\}$, respectively, acting as follows. The honest party and corrupted party choose their inputs uniformly at random; write x and y for the inputs chosen by P_1 and P_2. The adversary A^{P_1} chooses a round i uniformly at random. Then, before sending its messages for round i, if $\mathbf{a}_i \neq x$, the adversary aborts party P_1 without sending further messages and instructs Q to continue honestly with P_2; otherwise, it sends its messages for round i and aborts P_1 alone. The adversary A^{P_2} chooses a round i uniformly at random. Then, after sending its messages for round i, if $\mathbf{b}_i \neq y$, the adversary aborts P_2 without sending further messages and instructs Q to continue honestly with P_1; otherwise, it sends its messages for round $i+1$ and aborts P_2 alone. Adversary A^{P_1} outputs $\mathbf{b}_{i-1}$ or $\mathbf{b}_i$ (depending on the round P_1 aborted) and A^{P_2} outputs $\mathbf{a}_i$ or $\mathbf{a}_{i+1}$ (depending on the round P_2 aborted). We show that at least one of the adversaries outputs the honest party's input with probability noticeably greater than 1/3, in violation of privacy. Next, we compute each of the relevant probabilities.

$$\Pr\left[A^{P_1} \text{ outputs } y\right] = \frac{1}{r} \cdot \sum_{i=1}^{r} \left(\Pr_{\substack{x \leftarrow \mathbf{X} \\ y \leftarrow \mathbf{Y}}} \left[\mathbf{a}_i \neq x \wedge \mathbf{b}_{i-1} = y\right] + \Pr_{\substack{x \leftarrow \mathbf{X} \\ y \leftarrow \mathbf{Y}}} \left[\mathbf{a}_i = x \wedge \mathbf{b}_i = y\right] \right)$$

$$\Pr\left[A^{P_2} \text{ outputs } x\right] = \frac{1}{r} \cdot \sum_{i=0}^{r-1} \left(\Pr_{\substack{x \leftarrow \mathbf{X} \\ y \leftarrow \mathbf{Y}}} \left[\mathbf{b}_i \neq y \wedge \mathbf{a}_i = x\right] + \Pr_{\substack{x \leftarrow \mathbf{X} \\ y \leftarrow \mathbf{Y}}} \left[\mathbf{b}_i = y \wedge \mathbf{a}_{i+1} = x\right] \right)$$

Next, we compute the average of the two quantities above.

$$\left(\Pr\left[A^{P_1} \text{ outputs } y\right] + \Pr\left[A^{P_2} \text{ outputs } x\right]\right)/2 =$$
$$\frac{1}{2r}\left(\Pr_{\substack{x \leftarrow \mathbf{X} \\ y \leftarrow \mathbf{Y}}} \left[\mathbf{b}_0 \neq y \wedge \mathbf{a}_0 = x\right] + \Pr_{\substack{x \leftarrow \mathbf{X} \\ y \leftarrow \mathbf{Y}}} \left[\mathbf{a}_r = x \wedge \mathbf{b}_r = y\right] + \sum_{i=1}^{r-1} \Pr_{\substack{x \leftarrow \mathbf{X} \\ y \leftarrow \mathbf{Y}}} \left[\mathbf{a}_i = x\right] + \sum_{i=0}^{r-1} \Pr_{\substack{x \leftarrow \mathbf{X} \\ y \leftarrow \mathbf{Y}}} \left[\mathbf{b}_i = y\right] \right)$$

By correctness of the protocol and simplifying assumptions,

$$\begin{aligned} \left(\Pr\left[A^{P_1} \text{ outputs } y\right] + \Pr\left[A^{P_2} \text{ outputs } x\right]\right)/2 &= \frac{1}{2r} \cdot \Pr_{\substack{x \leftarrow \mathbf{X} \\ y \leftarrow \mathbf{Y}}} \left[\mathbf{b}_0 \neq y \wedge \mathbf{a}_0 = x\right] + \frac{1}{3} \\ &= \frac{1}{3} + \frac{1}{2r} \cdot \frac{2}{9} \end{aligned}$$

We conclude that at least one of the adversaries can guess with certainty the opponent's input with probability noticeably greater than 1/3, thus violating privacy.

3.2 General Case

We define a class Ω of 3-party functions, and we show that no function in this class admits a fully secure realization. Intuitively, this class of functions satisfies the following requirement: For both $\ell \in \{1,2\}$, there is a (non-trivial) partition of the inputs of P_ℓ and a distribution over the inputs of P_ℓ such that if P_ℓ samples its input according to the specified distribution then, with some fixed probability bounded away from 0 or 1, the output alone[5] fully determines what set of the partition P_ℓ's chosen input belongs to, *no matter* how the inputs of Q and $P_{3-\ell}$ were chosen. Furthermore, if both parties sample their inputs according to their respective distributions, then either for both inputs their sets in the partitions are determined from the output alone, or for neither. Formally,

Definition 3.3. *The class of functions Ω consists of all functions f satisfying the following conditions, for some $\gamma_1, \gamma_2 \in (0,1)$. There exist distributions $\mathbf{X}$ and $\mathbf{Y}$ over X and Y, respectively, such that $\operatorname{supp}(\mathbf{X}) = X$ and $\operatorname{supp}(\mathbf{Y}) = Y$, and partitions $X_1 \dots X_k$ and $Y_1 \dots Y_\ell$ of X and Y, respectively, such that*

1. *For every distribution Δ_1 over $X \times Z$,*
$$\Pr_{\substack{(x_0,z_0)\leftarrow\Delta_1\\ \widetilde{y}\leftarrow\mathbf{Y}}}\left[\exists j \text{ s.t. } \Pr_{y'\leftarrow\mathbf{Y}}\left[y' \in Y_j \mid f(x_0,\widetilde{y},z_0) = f(x_0,y',z_0)\right] = 1\right] \;=\; \gamma_1$$
2. *For every distribution Δ_2 over $Y \times Z$,*
$$\Pr_{\substack{\widetilde{x}\leftarrow\mathbf{X}\\ (y_0,z_0)\leftarrow\Delta_2}}\left[\exists j \text{ s.t. } \Pr_{x'\leftarrow\mathbf{X}}\left[x' \in X_j \mid f(\widetilde{x},y_0,z_0) = f(x',y_0,z_0)\right] = 1\right] \;=\; \gamma_2$$
3. *There exists $z_0 \in Z$ such that, for every $\sigma \in \Sigma$,*
$\exists j$ *s.t.* $\Pr\left[\widetilde{x} \in X_j \mid f(\widetilde{x},\widetilde{y},z_0) = \sigma\right] = 1$ *if and only if*
$\exists j$ *s.t.* $\Pr_{\substack{\widetilde{x}\leftarrow\mathbf{X}\\ \widetilde{y}\leftarrow\mathbf{Y}}}\left[\widetilde{y} \in Y_j \mid f(\widetilde{x},\widetilde{y},z_0) = \sigma\right] = 1$

Note that $\mathsf{PSI}^{\mathrm{id}}_{m,k}$, with $0 < k < m/2$, satisfies the above definition: define $\mathbf{X} = \mathbf{Y}$ as the uniform distribution and define partitions $\{X_x = \{x\}\}_{x\in X}$ and $\{Y_y = \{y\}\}_{y\in Y}$.

Remark 3.4. *The class of functions Ω can be generalized in few ways that we omitted, for the sake of presentation. The first generalization considers functions that take more than three inputs and can be reduced to functions in Ω by grouping parties together. The second generalization relaxes the requirement on the support of the distributions $\mathbf{X}$ and $\mathbf{Y}$ (allowing $\operatorname{supp}(\mathbf{X}) \subsetneq X$ or $\operatorname{supp}(\mathbf{Y}) \subsetneq Y$). The proof for the latter is almost identical to the one below.*

Theorem 3.5. *For any $f \in \Omega$ and for any protocol Π computing f, at least one of the following holds.*

- *There exists an adversary corrupting either P_1 or P_2 that can violate correctness.*
- *There exists an adversary corrupting either Q and P_1, or Q and P_2 that can violate privacy.*

Hereafter, fix a function f, real numbers $\gamma_1, \gamma_2 \in (0,1)$, distributions $\mathbf{X}$ and $\mathbf{Y}$ and partitions $X_1 \dots X_k$ and $Y_1 \dots Y_\ell$, and z_0 satisfying Definition 3.3. It is immediate that $\gamma_1 = \gamma_2$, hence we simply write γ $(= \gamma_1 = \gamma_2)$. We define $4r+1$ adversaries $\{A_i^{P_1}\}_{i=1}^{r}$, $\{A_i^{P_2}\}_{i=0}^{r-1}$, $\{C_i^{P_\ell}\}_{i=1}^{r}$ and $\widetilde{A}_0^{P_1}$ (See Figure 2). Let $\Sigma' \subset \Sigma$ denote all the elements $\sigma \in \Sigma$ such that there exists j for which $\Pr_{\substack{\widetilde{x}\leftarrow\mathbf{X}\\ \widetilde{y}\leftarrow\mathbf{Y}}}\left[\widetilde{y} \in Y_j \mid f(\widetilde{x},\widetilde{y},z_0) = \sigma\right] = 1$. Such a Σ' is guaranteed to exist by Item 2 of Definition 3.3.

[5] Without knowledge of the inputs of Q and $P_{3-\ell}$.

Adversaries $\{C_i^{P_\ell}\}_{i=1}^r$, $\widetilde{A}_0^{P_1}$, $\{A_i^{P_1}\}_{i=1}^r$ and $\{A_i^{P_2}\}_{i=0}^{r-1}$

- Each $C_i^{P_1}$ corrupts party P_1 and uses input sampled according to $\mathbf{X}$.

  Party P_1 aborts at round i.

- Each $C_i^{P_2}$ corrupts party P_2 and uses input sampled according to $\mathbf{Y}$.

  Party P_2 aborts at round i.

- Adversary $\widetilde{A}_0^{P_1}$ corrupts P_1 and Q and is *non-rushing.* The adversary instructs P_1 to use an input sampled according to $\mathbf{X}$ and Q to use input z_0. The adversary computes a_0 using the aforementioned inputs and P_1 and Q's random input. If $a_0 \in \Sigma'$, the adversary aborts P_1 alone without sending the first message and computes b_0 with the honest party. It outputs 1 if $b_0 \in \Sigma'$; otherwise, it continues honestly till the end and computes b_r with the honest party. The adversary outputs 1 if $b_r \in \Sigma'$.

- Each $A_i^{P_1}$ corrupts P_1 and Q and is *rushing.* The adversary instructs P_1 to use an input sampled according to $\mathbf{X}$ and Q to use input z_0. At round i, before sending the i-th round message, the adversary computes the backup value a_i and checks whether $a_i \in \Sigma'$ i.e. whether the backup value leaks its own input. If so, the adversary aborts P_1 alone at round $i+1$ and computes b_i with the honest party, or the adversary aborts P_1 alone at round i and computes b_{i-1} with the honest party. The adversary outputs 1 if $b_i \in \Sigma'$ (or $b_{i-1} \in \Sigma'$).

- Each $A_i^{P_2}$ corrupts P_2 and Q and is *non-rushing.* The adversary instructs P_2 to use an input sampled according to $\mathbf{Y}$ and Q to use input z_0. At round i, after sending the i-th round message, the adversary computes the backup b_i and checks whether $b_i \in \Sigma'$ i.e. he checks whether the backup leaks his own input. If so, the adversary aborts at round $i+2$ (i.e. sends one more message) and learns a_{i+1}, or the adversary aborts at round $i+1$ learns a_i. The adversary outputs 1 if $a_i \in \Sigma'$ (or $a_{i+1} \in \Sigma'$), and 0 otherwise.

Figure 2: Description of the Adversaries

Proof. Define $\widetilde{\mathbf{a}}_0, \ldots, \widetilde{\mathbf{a}}_r$ and $\widetilde{\mathbf{b}}_0, \ldots, \widetilde{\mathbf{b}}_r$ such that $\widetilde{\mathbf{a}}_i = 1$ (resp. $\widetilde{\mathbf{b}}_i = 1$) if and only if $\mathbf{a}_i \in \Sigma'$ (resp. $\mathbf{b}_i \in \Sigma'$) and 0 otherwise. In the following, we consider an execution of the protocol where Q uses z_0 as input, P_1 uses input sampled according to $\mathbf{X}$ and P_2 uses input sampled according to $\mathbf{Y}$, regardless of whether the parties are corrupted or not.

Claim 3.6. *Unless $C_i^{P_1}$ or $C_i^{P_2}$ violate correctness, it holds that* $|\Pr[\widetilde{\mathbf{b}}_i = 1] - \gamma|$, $|\Pr[\widetilde{\mathbf{a}}_i = 1] - \gamma| \leq \mathrm{neg}(\kappa)$*, for every $i \in \{0, \ldots, r-1\}$.*

Next, we analyze the probability that $A_i^{P_1}$ and $A_i^{P_2}$ output 1. Observe that, by correctness, with all but negligible probability, whenever $A_i^{P_1}$ (resp. $A_i^{P_2}$) outputs 1, the adversary succeeds in guessing the "bucket" the honest party's input belongs to, with certainty. To prove our theorem, we show that one of the adversaries $A_i^{P_\ell}$ or $\widetilde{A}_0^{P_1}$ outputs 1 with probability greater than γ, violating privacy.

$$\begin{aligned}\Pr\left[A_i^{P_1}\text{ outputs }1\right] &= \Pr\left[\widetilde{\mathbf{a}}_i = 0 \land \widetilde{\mathbf{b}}_{i-1} = 1\right] + \Pr\left[\widetilde{\mathbf{a}}_i = 1 \land \widetilde{\mathbf{b}}_i = 1\right]\\ \Pr\left[A_i^{P_2}\text{ outputs }1\right] &= \Pr\left[\widetilde{\mathbf{b}}_i = 0 \land \widetilde{\mathbf{a}}_i = 1\right] + \Pr\left[\widetilde{\mathbf{b}}_i = 1 \land \widetilde{\mathbf{a}}_{i+1} = 1\right]\end{aligned}$$

Therefore,

$$\begin{aligned}&\sum_{i=1}^{r}\Pr\left[A_i^{P_1}\text{ outputs }1\right] + \sum_{i=0}^{r-1}\Pr\left[A_i^{P_2}\text{ outputs }1\right] = \qquad (1)\\ &\quad \Pr\left[\widetilde{\mathbf{b}}_0 = 0 \land \widetilde{\mathbf{a}}_0 = 1\right] + \sum_{i=1}^{r-1}\Pr\left[\widetilde{\mathbf{a}}_i = 1\right] + \sum_{i=0}^{r-1}\Pr\left[\widetilde{\mathbf{b}}_i = 1\right] + \Pr\left[\widetilde{\mathbf{a}}_r = 1 \land \widetilde{\mathbf{b}}_r = 1\right]\end{aligned}$$

Thus

$$\sum_{i=1}^{r}\Pr\left[A_i^{P_1}\text{ outputs }1\right] + \sum_{i=0}^{r-1}\Pr\left[A_i^{P_2}\text{ outputs }1\right] = \Pr\left[\widetilde{\mathbf{b}}_0 = 0 \land \widetilde{\mathbf{a}}_0 = 1\right] + 2r\cdot\gamma \qquad (2)$$

The last equation follows by correctness and Items 1 to 3 of Definition 3.3. Next, we argue that $\Pr[\widetilde{\mathbf{b}}_0 = 0 \land \widetilde{\mathbf{a}}_0 = 1]$ is a noticeable quantity. If not, then we claim that adversary $\widetilde{A}_0^{P_1}$ can violate privacy. Suppose that $\Pr[\widetilde{\mathbf{b}}_0 = 0 \land \widetilde{A}_0^{P_1} = 1] \leq \mathsf{neg}(\kappa)$ and let ρ denote the (joint) randomness of parties P_1 and Q. In the presence of adversary $\widetilde{A}_0^{P_1}$, we claim that the events $\mathbf{a}_0 \notin \Sigma'$ and $\mathbf{a}_r \notin \Sigma'$ are independent of each other. To prove it, first notice that $\mathbf{a}_0$ may be viewed as deterministic function of the inputs of P_1 and Q and ρ, and $\mathbf{a}_r$ may be viewed as a deterministic function of the inputs of f (the latter assumption holds by correctness, with all but negligible probability). We write $\mathbf{a}_0(x, z_0; \rho)$ and $\mathbf{a}_r(x, y, z_0)$ to make the dependency explicit and compute:

$$\begin{aligned}&\Pr_{x\leftarrow\mathbf{X},y\leftarrow\mathbf{Y},\rho\leftarrow\mathbf{R}}\left[\mathbf{a}_0(x, z_0; \rho) \notin \Sigma' \land \mathbf{a}_r(x, y, z_0) \notin \Sigma'\right] =\\ &\quad \sum_{x_0\in X}\Pr_{y\leftarrow\mathbf{Y},\rho\leftarrow\mathbf{R}}\left[\mathbf{a}_0(x, z_0; \rho) \notin \Sigma' \land \mathbf{a}_r(x, y, z_0) \notin \Sigma' \mid x = x_0\right]\cdot\Pr_{x\leftarrow\mathbf{X}}\left[x = x_0\right]\end{aligned}$$

Observe that for any fixed x_0, the random variables $\mathbf{a}_0(x_0, y; \rho)$ and $\mathbf{a}_r(x_0, y, z_0)$ are independent random variables. Therefore,

$$\begin{aligned}&\Pr_{x\leftarrow\mathbf{X},y\leftarrow\mathbf{Y},\rho\leftarrow\mathbf{R}}\left[\mathbf{a}_0(x, z_0; \rho) \notin \Sigma' \land \mathbf{a}_r(x, y, z_0) \notin \Sigma'\right] =\\ &\quad \sum_{x_0\in X}\Pr_{\rho\leftarrow\mathbf{R}}\left[\mathbf{a}_0(x, z_0; \rho) \notin \Sigma' \mid x = x_0\right]\cdot\Pr_{y\leftarrow\mathbf{Y}}\left[\mathbf{a}_r(x, y, z_0) \notin \Sigma' \mid x = x_0\right]\cdot\Pr_{x\leftarrow\mathbf{X}}\left[x = x_0\right]\end{aligned}$$

Finally, by correctness and Item 2 of Definition 3.3

$$\begin{aligned}&\Pr_{x\leftarrow\mathbf{X},y\leftarrow\mathbf{Y},\rho\leftarrow\mathbf{R}}\left[\mathbf{a}_0(x, z_0; \rho) \notin \Sigma' \land \mathbf{a}_r(x, y, z_0) \notin \Sigma'\right]\\ &\quad = \sum_{x_0\in X}\Pr_{\rho\leftarrow\mathbf{R}}\left[\mathbf{a}_0(x, z_0; \rho) \notin \Sigma' \mid x = x_0\right]\cdot(1-\gamma)\cdot\Pr_{x\leftarrow\mathbf{X}}\left[x = x_0\right]\\ &\quad = (1-\gamma)\cdot\Pr_{x\leftarrow\mathbf{X},\rho\leftarrow\mathbf{R}}\left[\mathbf{a}_0(x, z_0; \rho) \notin \Sigma'\right] = (1-\gamma)^2\end{aligned}$$

The last equality follows from correctness and Item 1 of Definition 3.3. Thus, if $\Pr[\widetilde{\mathbf{b}}_0 = 0 \land \widetilde{\mathbf{a}}_0 = 1] \leq \mathrm{neg}(\kappa)$, then adversary $\widetilde{A}_0^{P_1}$ outputs 1 with probability $1-(1-\gamma)^2 > \gamma$, in violation of privacy. In conclusion, using an averaging argument in Equation (2), at least one of $\{A_i^{P_1}\}_{i=1}^{r}$, $\{A_i^{P_2}\}_{i=0}^{r-1}$ outputs 1 with probability noticeably greater than γ and, thus, violates privacy. □

4 Positive Results

In this section, we present our positive results. First, we give a generic transformation from a fully secure n-party protocol with non-solitary output to a fully secure $(n+1)$-party protocol with solitary output; The latter protocol computes the associated functionality that delivers output to an additional auxiliary party that doesn't provide input. In light of the positive results for fair two-party computation, our transformation enables fully secure computation for (almost all) Boolean functions with unequal domain size. For instance, it yields a secure protocol for the following PSI variant that escapes our other criteria: From a universe of size n, party P_1 picks a set of size between 1 and k, for some arbitrary fixed $k \leq n-2$, party P_2 picks a set size between 1 and $k+1$ (i.e. one party has more inputs to pick than the other), and Party Q receives value 1 if the sets intersect and 0 if not.[6] Interestingly, this technique yields protocols with super-constant (in fact, super-logarithmic) round complexity since, with few exceptions, super-logarithmic number of rounds is necessary for fair computation. In Section 5, we show that super-constant round complexity is inherent for fully secure MPC with solitary output.

Then, we present a generic protocol for functionalities that satisfy the "forced output distribution" criterion. Intuitively, these are functionalities where (almost) all parties can "force" the distribution of the output to be invariant of the other parties' choice of input. These functionalities should be contrasted with the above fair ones, since they are utterly unfair viewed as non-solitary functionalities (they imply coin-tossing). Interestingly, every functionality in this class can be computed in a constant number of rounds.

We also present a generic protocol for functionalities that satisfy the "fully revealing input" criterion. Intuitively, these are functionalities where at least one party has a choice of input that reveals all other parties' inputs. While this family may appear somewhat pathological from a cryptographic point of view, it contains several natural examples. In particular, it contains a PSI variant where one party may choose the entire universe as input. Similarly to the previous case, every functionality in this class can be computed in constant number of rounds.

Finally, for a functionality that escapes the above criteria, we design a fully secure protocol that runs in superlogarithmic number of rounds. This protocol is inspired by the GHKL protocol [23]. We emphasize that the feasibility of this functionality does *not* follow from the fairness criterion since, viewed as a non-solitary functionality, it cannot be computed fairly. Furthermore, in the next section, we show that superconstant round complexity is inherent for this function.

4.1 Security via Fairness

Let $f : X_1 \times \ldots \times X_n \to \Sigma$ be an n-party functionality that delivers the same output to all parties. Let Π be a fully secure protocol for f. Write $m_i^{(\ell,\ell')} \in \{0,1\}^{\mu_\kappa}$ for the message sent by P_ℓ to $P_{\ell'}$ at round i. Let $M_\kappa = \mu_\kappa \cdot n$ denote the total length of messages received by party P_ℓ in a single round (without loss of generality μ_κ and M_κ do not depend on i, ℓ' or ℓ). In this section, we show how to transform protocol Π into a protocol Π' that computes the associated solitary functionality that delivers the output to one of the parties, or to an additional auxiliary party. We note that the transformation and analysis of the two cases are the same, therefore we only focus on the latter transformation (i.e. from n-party to $n+1$-party protocol, where the output receiving party does not provide input). The rest of this sub-section is dedicated to the proof of the following theorem.

[6] Viewed as a two-party non-solitary functionality, the fact that it can be computed with full security (fairness) follows from the criteria of [3].

Theorem 4.1. *Let Π be a protocol for computing non-solitary functionality f with full security. Then, there exist a protocol Π' that computes with full security the associated $(n+1)$-party solitary functionality that delivers the output to an additional auxiliary party.*

At a high level, to transform the n-party non-solitary protocol Π into an $(n+1)$-party solitary protocol Π', we have each party P_ℓ in Π' share the view of the party P_ℓ in the original protocol Π between himself and the auxiliary party Q. To do so, we begin by defining protocol's Π message function NxtMsg_Π that deterministically maps each party P_ℓ's view until some round i (a view that includes its identity, its input, its private coins and all incoming messages until that round) to all messages that P_ℓ sends at the upcoming round.

Definition 4.2. *Let NxtMsg_Π denote the next message function of r-round protocol Π. Formally, NxtMsg_Π maps $\mathsf{view}_i^{P_\ell} \mapsto (m_{i+1}^{(\ell,1)}, \ldots, m_{i+1}^{(\ell,n)})$ such that*

1. *$\mathsf{view}_i^{P_\ell} \in \{0,1\}^{i \cdot M_\kappa}$ corresponds to the view of party P_ℓ up to and including round i (wlog, we assume that the value of i and the identity of P_ℓ are contained in its view).*

2. *If $i \neq r$, then $m_{i+1}^{(\ell,\ell')} \in \{0,1\}^{\mu_\kappa}$ corresponds to P_ℓ's prescribed message to $P_{\ell'}$ at round $i+1$ according to Π. If $i = r$ then $m_{i+1}^{(\ell,\ell')} \in \{0,1\}^{\mu_\kappa}$ corresponds to P_ℓ's prescribed output.*

In our protocol design, all messages will be additively-shared between party P and a helper party Q. That is, a message m will be randomly split into m_1, m_2 such that $m = m_1 \oplus m_2$ and party P will hold m_1 and Q will hold m_2. In the following functionality $\mathsf{ShrNxtMsg}_\Pi$ (Figure 3) we describe how the messages of the protocol are created to deliver this sharing. Party P and Q hold view_i^P, P's view up to and including round i, in shared form as v_P, v_Q and they receive the next round messages of P also in shared form.

Functionality $\mathsf{ShrNxtMsg}_\Pi$

- **Input:** Party P holds $v_P \in \{0,1\}^{i \cdot M_\kappa}$ and party Q holds $v_Q \in \{0,1\}^{i \cdot M_\kappa}$, $v_P \oplus v_Q = \mathsf{view}_i^P$.
- **Output:**
  1. **Case $i \neq r$.** Party P receives random $m_1 \leftarrow \{0,1\}^{M_\kappa}$ and Q receives $m_2 = m_1 \oplus \mathsf{NxtMsg}(v_P \oplus v_Q)$.
  2. **Case $i = r$.** Party P receives $m_1 = 0^{M_\kappa}$ and Q receives $m_2 = \mathsf{NxtMsg}(v_P \oplus v_Q)$.

Figure 3: Two-Party Functionality $\mathsf{ShrNxtMsg}_\Pi$ for Parties P and Q.

We describe the protocol for computing a function with an auxiliary party Q that receives the solitary output. The idea is that each party P_ℓ will invoke with party Q the protocol for creating the messages that P_ℓ needs to send to all the other parties in the upcoming round. This is done by utilizing the functionality $\mathsf{ShrNxtMsg}_\Pi$. The result is that P_ℓ and Q receive the set of messages $(m_{i+1}^{(\ell,1)}, \ldots, m_{i+1}^{(\ell,n)})$ in shared form. Then, P_ℓ send to each other party P_j its share of the message $m^{(\ell,j)}$. The auxiliary party Q holds in a string $\mathsf{view}_i^{Q_\ell}$ its share of the view of the messages of party P_ℓ up to and including round i (a different string for each P_ℓ). If (some) parties abort, then proceed under the specifications of the original protocol Π, while maintaining the invariant that each P_ℓ's view from the original protocol is shared between P_ℓ and Q. At the end of the execution, Q together with one of the P_ℓ's that hasn't aborted reconstruct the output (which is a deterministic function of their joint views).

$(n+1)$-Party Solitary Protocol from n-party Non-Solitary Protocol

Input: Each party P_ℓ holds input x_ℓ, random input ρ_ℓ and security parameter 1^κ. Party Q holds security parameter 1^κ and does not hold any private input.

Protocol:

1. Q sets $\{\mathsf{view}_0^{Q_\ell} = 0^{M_\kappa}\}_{\ell\in[n]}$ and every other party sets $\mathsf{view}_0^{P_\ell} = (1^\kappa, x_\ell, \rho_\ell)$
2. For $i = 1, \ldots, r$
   (a) Each P_ℓ and Q invoke $\mathsf{ShrNxtMsg}_\Pi$ on input $\mathsf{view}_{i-1}^{Q_\ell}$ and $\mathsf{view}_{i-1}^{P_\ell}$.
   (b) Each party P_ℓ (other than Q) is instructed to send his share of message $m_i^{\ell,\ell'}$ to $P_{\ell'}$.
   (c) Each P_ℓ (other than Q) computes $\mathsf{view}_i^{P_\ell}$ by concatenating $\mathsf{view}_{i-1}^{P_\ell}$ with the (shares of the) messages they received at Step 2b. Party Q computes each $\mathsf{view}_i^{Q_\ell}$ by concatenating $\mathsf{view}_{i-1}^{Q_\ell}$ with the (shares of the) messages he received from the invocations at Step 2a (collating them appropriately).

   Abort scenarios:

   - If Q aborts, then all parties halt.
   - If any of the P_j abort, then each remaining P_ℓ and Q update $\mathsf{view}_i^{P_\ell}$ and $\mathsf{view}_i^{Q_\ell}$, respectively, such that the latter jointly reflect that P_j stopped sending messages from that round onward (since the original protocol Π is fully secure, the remaining parties will be able to continue with the execution).
3. Initialize $\mathsf{ctr} = 1$.
   (a) As long as $\mathsf{ctr} \leq n$ and P_{ctr} is not alive or aborted, then increment ctr.
   (b) If $\mathsf{ctr} \leq n$, then P_{ctr} and Q invoke $\mathsf{ShrNxtMsg}_\Pi$ on input $\mathsf{view}_r^{Q_{\mathsf{ctr}}}$ and $\mathsf{view}_r^{P_{\mathsf{ctr}}}$. Party Q outputs whatever he receives from the invocation, and notifies all parties to halt. Otherwise, if $\mathsf{ctr} = n+1$, then Q outputs $f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ where $(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ is chosen uniformly at random from $X_1 \times \ldots \times X_n$.

Output: Party Q's output is determined at Step 3b. Other parties output nothing.

Figure 4: $(n+1)$-Party Protocol for Solitary f in the $\mathsf{ShrNxtMsg}_\Pi$-Hybrid Model with Identifiable Abort

The above protocol is described where the output is delivered to the auxiliary party Q (not one of the $P_1 \dots P_n$). However, as noted at the beginning of this section, this party can be one of the n original parties and simply serves both as himself and as party Q. Observe that, in this case, Q will simply see all the messages that it sends and receives (as it holds both shares of the messages).

Proof of Theorem 4.1. We prove the claim by showing that protocol Π' from Figure 4 is fully secure in the $\mathsf{ShrNxtMsg}_{\Pi}$-hybrid model with identifiable abort. Then, the theorem follows from composition [8]. Let A be an adversary corrupting up to n parties (of the $n+1$ parties). Observe that, if party Q is not among the corrupted parties, then A's view can be trivially simulated since it is just a uniform random string, and it is not hard to see that he cannot affect correctness. It remains to prove that the protocol is secure when Q is among the corrupted parties. Let $\mathcal{C}$ denote the set of corrupt parties, assuming that $Q \in \mathcal{C}$. For adversary A attacking Π' corrupting parties in $\mathcal{C}$, we construct an adversary $\widetilde{A}$ attacking Π (on the same input distribution and auxiliary information) and corrupting parties $\widetilde{\mathcal{C}} = \mathcal{C} \setminus \{Q\}$ (there are at most $n-1$ such parties). Since A's and $\widetilde{A}$'s views are identically distributed (modulo a 2-out-of-2 secret sharing), and since the latter can be simulated in the ideal model with full security, it follows that the former can be simulated as well. Formally, let $\widetilde{S}$ denote the simulator for $\widetilde{A}$ and define simulator S for A as follows:

1. S runs $\widetilde{S}$ on the relevant inputs, security parameter and auxiliary information. Write $\left(v_{P_i}\right)_{P_i \in \widetilde{\mathcal{C}}}$ for $\widetilde{S}$'s output corresponding to the joint simulated view of the parties.
2. S samples $\left(\nu_{P_i}\right)_{P_i \in \widetilde{\mathcal{C}}}$ uniformly at random from the relevant space and outputs $\left(v_{P_i} \oplus \nu_{P_i}\right)_{P_i \in \widetilde{\mathcal{C}}}$ (the simulated views of parties in $\widetilde{\mathcal{C}}$) and $\left(\nu_{P_i}\right)_{P_i \in \widetilde{\mathcal{C}}}$ (the simulated view of Q).

□

4.2 Functions with Forced Output Distribution

In this section, we present the "Forced Output Distribution" criterion. First, we define the notion.

Definition 4.3. *A party $P_i \neq Q$ admits a* forced output distribution *for f if there exists a distribution Δ_i over X_i such that the distribution of the random variable $f(x_1, \dots, x_{i-1}, \hat{x}_i, x_{i+1}, \dots, x_n)|_{\hat{x}_i \leftarrow \Delta_i}$ is independent of the $(n-1)$-tuple $(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$.*

Intuitively, a party admits a *forced output distribution* if it can choose its input in a way that "forces" the output, i.e. it makes the output distribution independent of the other parties' inputs. The theorem below states that if all-but-one parties, not including Q, admit a forced output distribution, then the functionality is computable with perfect full security in a constant number of rounds in a hybrid model with ideal access with identifiable abort to functionality ShrGn_f (to be specified below). As a corollary, assuming OT, functions with a forced output distribution admit fully secure protocol in the plain model.

Theorem 4.4. *Assume that at least $n-1$ of the parties in $\mathcal{P} \setminus \{Q\}$ admit a forced output distribution for functionality f. Then, f is computable with perfect full security in the ShrGn_f-hybrid model with identifiable abort. Furthermore, the computation runs in a constant number of rounds.*

We now introduce functionality ShrGn_f (Figure 5) and we will prove our theorem in the ShrGn_f-hybrid model with identifiable abort. This functionality provides the following. It shares the output of the function f between the parties that invoke it, by obliviously choosing a random input for the parties that do not provide input. That is, it provides uniform random shares to all-parties-but-one, and that last party gets the *xor* of these shares with the output of the function. We emphasize that this functionality may be invoked by a *subset* of the n parties, and, as per the ideal model with identifiable abort, the invocation can be aborted by any single party in that set (at the cost of revealing its identity).

Functionality ShrGn_f

- **Input:** Each $P_i \in \mathcal{P}$ provides input x_i.
- **Computation:**
  1. If parties in $\mathcal{T}$ do not provide input then $x_{\mathcal{T}} \leftarrow \bigtimes_{P_i \in \mathcal{T}} X_i$ is chosen uniformly at random.

     Write $S_1, \ldots, S_\ell$ for the parties in $\mathcal{S} = \mathcal{P} \setminus \mathcal{T}$.
  2. Compute $\left\{\sigma_{S_j} \in \Sigma\right\}_{S_j \in \mathcal{S}}$ such that
     (a) $\sigma_{S_j} \leftarrow \Sigma$ independently and uniformly at random, for each $S_j \neq S_1$ (i.e., $2 \leq j \leq \ell$).
     (b) $\sigma_{S_1} = f(x_{\mathcal{S}}, x_{\mathcal{T}}) \oplus \sigma_{S_2} \oplus \ldots \oplus \sigma_{S_\ell}$.
- **Output:** Each $S_j \in \mathcal{S}$ receives σ_{S_j}.

Figure 5: n-Party Functionality ShrGn_f.

Without loss of generality, if it exists, suppose that P_1 is the party without forced output distribution (the protocol and our analysis remains sound if all parties have a forced output distribution). The protocol (see Figure 6) proceeds as follows: the parties invoke the trusted party for computing ShrGn_f, and obtain shares of the output. Then, in two distinct steps (1) P_1 sends its share of the output to Q and (2) all other parties send their shares to Q. In case of abort, there are two scenarios; either P_1 aborts alone, in which case the process starts again without P_1, or, if anyone else aborts at this iteration or the next, the computation halts and Q outputs a value from the forced distribution. Intuitively, the protocol maintains security because it is not useful to abort any of the parties; aborting any party but P_1 halts the execution, while aborting P_1 does not reveal anything about the output (since the honest party will not send its share before P_1 sends his).

Proof of Theorem 4.4. First note that distribution $\mathbf{D}$ in Figure 6 is well defined since it is unique. Let A denote an adversary corrupting a subset of parties. Like in the previous proof, it is straightforward that if A does not corrupt Q then it cannot affect correctness and its view can be trivially simulated. Let $\mathcal{C}$ be the set of corrupted parties. Define simulator S that does the following: S invokes the trusted party on the inputs of the corrupted parties and receives output out form the trusted party. Then, S samples $|\mathcal{C}|$ random elements $\{\sigma'_C\}_{C \in \mathcal{C}}$ and hands them to the adversary.

- If P_1 alone aborts, S samples $|\mathcal{C}| - 1$ fresh random values $\{\sigma''_C\}_{C \in \mathcal{C} \setminus \{P_1\}}$, and hands them to the adversary.

- If any other party aborts (at any point in the simulation), S samples $d' \leftarrow \mathbf{D}$, hands d to the adversary, and outputs whatever A outputs.
- If no other party aborts, S hands out to the adversary and outputs whatever A outputs. □

n-Party Protocol Π for Computing f with Forced Output Distribution

Recall that P_1 denotes the party that does not admit a forced output distribution (if there is such party). Write $\mathbf{D}$ for the distribution induced by (any of) the forced distributions.

Input: Each party P_ℓ holds input x_ℓ (Recall that Q is one of the P_ℓ's).

Protocol:

1. Set $\mathcal{S} = \mathcal{P}$.
2. Invoke the oracle computing ShrGn_f. If parties in $\mathcal{T} \subset \mathcal{S}$ abort the computation, then the parties set $\mathcal{S} = \mathcal{S} \setminus \mathcal{T}$ and repeat the current step. If the computation is not aborted then each $P \in \mathcal{S}$ receives σ_P.
3. If P_1 is still alive,

   (a) P_1 sends σ_{P_1} to Q.

   (b) Parties in $\mathcal{S} \setminus \{P_1\}$ send their shares to Q.

   Abort scenarios:

   - If Q aborts, then the computation halts.
   - If P_1 aborts at Step 3a, then the parties set $\mathcal{S} = \mathcal{S} \setminus \mathcal{T}$ and go back to Step 2.
   - If parties in $\mathcal{T} \subseteq \mathcal{S} \setminus \{Q, P_1\}$ abort, then Q outputs $d \leftarrow \mathbf{D}$ and the computation halts.
4. If P_1 already aborted,

   (a) Parties in $\mathcal{S}$ send their shares to Q.

   Abort scenarios:

   - If Q aborts, then the computation halts.
   - If parties in $\mathcal{T} \subseteq \mathcal{S} \setminus \{Q\}$ abort the computation, then Q outputs $d \leftarrow \mathbf{D}$ and the computation halts.

Output: If Q's output hasn't been determined yet, then Q outputs $\bigoplus_{P \in \mathcal{S}} \sigma_P$. All other parties output nothing.

Figure 6: n-Party Protocol Π for f with Ideal Access to ShrGn_f with Identifiable Abort

4.3 Functions with Fully Revealing Input

In this section, we present the "Fully Revealing Input" criterion. First, we define the notion.

Definition 4.5. *Let $\mathcal{S} \subsetneq \mathcal{P}$. We say that the parties in $\mathcal{S}$ admit a fully revealing input, if there exists $x_{\mathcal{S}} \in \underset{P_i \in \mathcal{S}}{\times} X_i$ such that the following function is injective*

$$f_{x_{\mathcal{S}}} : x_{\overline{\mathcal{S}}} \mapsto f(x_{\mathcal{S}}, x_{\overline{\mathcal{S}}}).$$

The theorem below states that if there exists a fixing of the inputs of P_1 and Q (or any P_i and Q) that yields an injective function, then the overlying functionality f is computable with full security in a constant number of rounds in the ShrGn_f-hybrid model. Similarly to the previous section, assuming OT, it follows as an immediate corollary that functions with fully revealing input admit fully secure protocol in the plain model.

Theorem 4.6. *Assume there exists i such that $\{P_i, Q\}$ admit a fully revealing input. Then, functionality f is computable with perfect full security in the ShrGn_f-hybrid model with identifiable abort. Furthermore, the computation runs in a constant number of rounds.*

Without loss of generality, suppose that P_1, Q admit a fully revealing input. The protocol (Figure 7) proceeds as follows: the parties invoke the trusted party for computing ShrGn_f, and obtain shares of the output. Then, in two distinct steps (1) All-parties-but-P_1 send their shares of the output to Q and (2) P_1 sends its share to Q. In case of abort, the process is repeated until it succeeds. Intuitively, the protocol maintains security because the only way to extract more information from the protocol is to corrupt both P_1 and Q. In that case however, P_1 and Q can provide input in the ideal model that reveals everything about the inputs of the honest parties.

Proof of Theorem 4.6. Let A denote the adversary corrupting a subset of parties. Like in the previous proof, it is straightforward that if A does not corrupt both Q and P_1 then it cannot affect correctness and its view can be trivially simulated. If A corrupts both P_1 and Q, then by instructing the simulator to send the fully revealing input in the ideal model, the adversary's view can be simulated perfectly, no matter what is its course of action.[7] □

4.4 Outliers

In this section, we present protocol for a function that escapes the above criteria but is nevertheless computable with full security. Define functionality f that takes inputs $x \in \{0, 1, 2\}$ from P_1 and $y \in \{0, 1, 2\}$ from P_2 and delivers $f(x, y)$ to Q such that

$$f(x,y) = \begin{cases} 1 & \text{if } x = y \in \{0,1\} \\ 2 & \text{if } x = y = 2 \\ 0 & \text{otherwise} \end{cases}$$

In this section, we show that the functionality f is computable with full security in $\omega(\log(\kappa))$ rounds. In what follows, we identify $\{0, 1, 2\}$ with $\{x_0, x_1, x_2\}$ or $\{y_0, y_1, y_2\}$ to make the distinction between

[7] We stress that honest-but-curious adversaries can be simulated without having recourse to the fully revealing input, conforming to the standard definition. We have omitted the analysis here, since it is straightforward.

n-Party Protocol for Computing f with Fully Revealing Input

Recall that P_1 denotes the party such that $\{P_1, Q\}$ admit a fully revealing input.

- **Input:** Each party P_ℓ holds input x_ℓ (recall that here Q is one of the P_ℓ's).

1. The parties invoke the oracle computing ShrGn_f and each $P_i \in \mathcal{S}$ receives σ_{P_i}.

   Abort scenarios:

   - If Q aborts, then the computation halts.
   - If parties in $\mathcal{T} \subsetneq \mathcal{S} \setminus \{Q\}$ abort the computation, then the remaining parties set $\mathcal{S} = \mathcal{S} \setminus \mathcal{T}$ and repeat the present step.
   - If parties in $\mathcal{T} = \mathcal{S} \setminus \{Q\}$ abort the computation, then Q outputs $f(x_{\{Q\}}, x_\mathcal{T})$ where $x_\mathcal{T} \leftarrow \bigtimes_{P_i \neq Q} X_i$ is sampled uniformly at random.

2. If P_1 is still alive then

   (a) All parties except P_1 send their shares to Q.

   (b) P_1 sends σ_{P_1} to Q.

   Abort scenarios:

   - If Q aborts, then the computation halts.
   - If parties in $\mathcal{T} \subsetneq \mathcal{S} \setminus \{Q\}$ abort at Steps 3a or 2a, then the parties update $\mathcal{S} = \mathcal{S} \setminus \mathcal{T}$ and go back to Step 1.
   - If parties in $\mathcal{T} = \mathcal{S} \setminus \{Q\}$ abort the computation, then Q outputs $f(x_{\{Q\}}, x_\mathcal{T})$ where $x_\mathcal{T} \leftarrow \bigtimes_{P_i \neq Q} X_i$ is sampled uniformly at random.

3. If P_1 already aborted then

   (a) Remaining parties send their shares to Q.

   Abort scenarios:

   - If Q aborts, then the computation halts.
   - If parties in $\mathcal{T} \subsetneq \mathcal{S} \setminus \{Q\}$ abort at Step 2a, then the parties update $\mathcal{S} = \mathcal{S} \setminus \mathcal{T}$ and go back to Step 1.
   - If parties in $\mathcal{T} = \mathcal{S} \setminus \{Q\}$ abort the computation, then Q outputs $f(x_{\{Q\}}, x_\mathcal{T})$ where $x_\mathcal{T} \leftarrow \bigtimes_{P_i \neq Q} X_i$ is sampled uniformly at random.

- **Output:** If Q's output hasn't been determined yet, then Q outputs $\bigoplus_{P_i \in \mathcal{S}} \sigma_{P_i}$. All other parties output nothing.

Figure 7: n-Party Protocol Π for f in the ShrGn_f-Hybrid Model with Identifiable Abort

the parties' input-spaces explicit. For readability, it is convenient to represent the function f by its truth table:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

Our protocol is inspired by the GHKL protocol and proceeds as follows. Formal descriptions and more detailed security analysis appear in Appendix B. In the remainder of this section, we only give a high level overview. Write x and y for the inputs used by the parties. In a share-generation phase, the parties obliviously generate two sequences of values $(a_0, \dots, a_r)$ and $(b_0, \dots, b_r)$ and an integer $i^* \in [r]$ such that every value a_i and b_i is equal to $f(x, y)$ for indices succeeding i^*, and, for indices preceding i^*, a_i is computed by obliviously choosing a fresh input from $\{y_0, y_1\}$ for P_2, and using input x for P_1 and, similarly, b_i is computed by obliviously choosing a fresh input from $\{x_0, x_1\}$ for P_1, and using input y for P_2. The value of i^* is chosen according to a distribution specified in Figure 9. The two sequences are then shared in a 3-out-of-3 additive (modulo 3) secret sharing among the parties. Then, in the share-exchange phase, in r iterations, P_1 is instructed to send its share of b_i to Q, and P_2 is instructed to send its share of a_i to Q. If party P_1 aborts at round i, then P_2 sends its share of b_{i-1} to Q, and, similarly, if P_2 aborts at round i, then P_1 sends its share of a_i to Q. Party Q is instructed to output the value it can reconstruct from the shares. The share-generation and share-exchange phases are fully specified in Figures 9 and 10, respectively, in Appendix B.

We crucially observe that, prior to i^*, the obliviously chosen input for each party is sampled from $\{0, 1\}$, and *not* $\{0, 1, 2\}$. This seemingly superficial technicality is what enables the protocol to be secure, as we outline next.

High-level Security of Protocol Π from Figure 10. The only interesting case for security is what happens when P_1 and Q are both corrupt (and the symmetric case when P_2 and Q are corrupt that is handled analogously). It is straightforward that all other corruption configuration pose no potential threat to the protocol's security. We only focus here on the interesting. Let A denote the adversary and consider the following simulator S: The simulator samples i^* according to the prescribed distribution, and reveals (shares of) the sequence $a_0, a_1 \dots a_{i^*-1}$ in $i^* - 1$ iterations; then, at iteration i^*, the simulator invokes the trusted party to learn the correct output and proceeds in $r - i^* + 1$ iterations revealing (appropriate fresh shares of) $f(x, y)$. If the adversary decides to abort at round i, then the simulator computes (appropriate shares of) a bit b_{i-1} which it hands to the adversary. The simulator then outputs whatever the adversary outputs and halts. The crux of our security proof lies in carefully choosing the value of b_{i-1}. The simulation is specified in Figure 11 (in Appendix B). On the surface, it appears that the adversary has an advantage in the real model by guessing i^* correctly (which happens with constant probability). In fact, this advantage can be simulated in the ideal model since, when the adversary guesses the round wrongly (and thus fails), the simulator can choose a different input than the one the adversary uses. Thus, on average, the simulator in the ideal world and the adversary in the real world have the same amount of information about the honest party's input.

We conclude with the following theorem which immediately yields full security for f, assuming a protocol for OT.

Theorem 4.7. *Protocol Π computes f with statistical full security in the ShrGn^*_f-hybrid model with identifiable abort.*

5 Lower-Bound on Round-Complexity

In this section, we present a lower bound for the functionality f from the previous section. Let f be the three-party solitary functionality from Section 4.4. In what follows, let Π denote a protocol for f, let κ denote the security parameter, and assume the round-complexity of Π is set to some value r that is independent of κ. It follows as an immediate corollary of the theorem below that no such protocol can be fully secure.

Theorem 5.1. *Using the notation above, there exists $i \in [r]$ such that at least one of the following is true:*

1. *An adversary corrupting P_2 and Q violates P_1's privacy by aborting P_2 at round i.*

2. *An adversary corrupting P_1 and Q violates P_2's privacy by aborting P_1 at round i.*

3. *An adversary corrupting P_1 violates correctness by aborting at round i.*

4. *An adversary corrupting P_2 violates correctness by aborting at round i.*

Proof. Following Notation 3.2, write $\mathbf{a}_0 \dots \mathbf{a}_r$ and $\mathbf{b}_0 \dots \mathbf{b}_r$ for the backup values, i.e. $\mathbf{a}_i$ (resp. $\mathbf{b}_i$) is the random variable representing Q output if P_2 (resp. P_1) does not send the i-th message, and P_2 and Q (resp. P_1 and Q) follow the protocols specifications. Let i denote the largest index such that $\Pr[\mathbf{a}_i \neq \mathbf{b}_{i-1}]$ or $\Pr[\mathbf{a}_{i-1} \neq \mathbf{b}_i]$ is a noticeable quantity. Since the protocol constant round, the value of i is well defined. Without loss of generality assume that $\Pr[\mathbf{a}_i \neq \mathbf{b}_{i-1}] \geq \varepsilon$ for some non-negligible ε. In what follows, all probabilities are computed over the parties' random inputs and choice of input for the functionality. To make the dependency explicit, we will write $\mathbf{a}_i(x, y)$ and $\mathbf{b}_i(x, y)$. We will omit mention of the random input. The claim follows from the fact that $\Pr[\mathbf{a}_i \neq \mathbf{b}_{i-1}] \geq \varepsilon$ and Claims 5.2 to 5.4.

Claim 5.2. *For every $(x, y) \in X \times Y$, it holds that $\Pr[\mathbf{a}_i(x, y) \neq f(x, y)] \leq \operatorname{neg}(\kappa)$.*

The above claim follows immediately from the definition of index i, and Π's correctness. Next, we define the following adversaries $A^{P_1}(x)$, $A^{P_2}(x)$ and $C^{P_1}(x)$ and $C^{P_2}(y)$.

1. Adversary $A^{P_1}(x)$ corrupts P_1 and Q, and uses input x for the computation. The adversary aborts P_1 at round i.

2. Adversary $A^{P_2}(y)$ corrupts P_2 and Q, and uses input y for the computation. The adversary aborts P_2 at round $i + 1$.

3. Adversary $C^{P_1}(x)$ corrupts P_1 alone, and uses input x for the computation. The adversary aborts P_1 at round i.

4. Adversary $C^{P_2}(y)$ corrupts P_2 alone, and uses input y for the computation. The adversary aborts P_2 at round $i + 1$.

Claim 5.3. *Suppose there exists $x \in \{x_0, x_1\}$ and $y \in Y$ such that $\Pr[\mathbf{a}_i(x, y) \neq \mathbf{b}_{i-1}(x, y)] \geq 1/\operatorname{poly}(\kappa)$, then $A^{P_1}(x)$ violates privacy, or $C^{P_1}(x)$ violates corectness, for some $x \in \{x_0, x_1\}$.*

Proof of Claim 5.3. Without loss of generality, suppose that $\Pr_{y\leftarrow Y}\left[\mathbf{a}_i(x_0,y) \neq \mathbf{b}_{i-1}(x_0,y)\right] \geq 1/\operatorname{poly}(\kappa)$. We proceed under the assumption that $C^{P_1}(x)$ does not violate correctness. First, notice that $\mathbf{a}_i$ and $\mathbf{b}_{i-1}$ cannot be equal to distinct nonzero values, by correctness. Therefore, at least one of the following occurs with noticeable probability:

1. $\Pr_{y\leftarrow Y}\left[\mathbf{a}_i(x_0,y) = 1 \wedge \mathbf{b}_{i-1}(x_0,y) = 0\right]$
2. $\Pr_{y\leftarrow Y}\left[\mathbf{a}_i(x_0,y) = 0 \wedge \mathbf{b}_{i-1}(x_0,y) = 1\right]$
3. $\Pr_{y\leftarrow Y}\left[\mathbf{a}_i(x_0,y) = 0 \wedge \mathbf{b}_{i-1}(x_0,y) = 2\right]$

If either of Items 2 and 3 is noticeable, then $A(x_0)$ violates privacy in an execution where P_2 chooses his input at random (sinbe $\mathbf{b}_{i-1}(x_0,y)$ and x_0 together determine y). If only the first item is noticeable, observe that

$$\begin{aligned}&\Pr_{y\leftarrow Y}\left[\mathbf{a}_i(x_0,y) = 0 \wedge \mathbf{b}_{i-1}(x_0,y) = 0\right] = \\ &\qquad\qquad\Pr\left[\mathbf{a}_i(x_0,y) = 0\right] - \Pr\left[\mathbf{a}_i(x_0,y) = 0 \wedge \mathbf{b}_{i-1}(x_0,y) = 1\right].\end{aligned}$$

and thus $\Pr_{y\leftarrow Y}\left[\mathbf{a}_i(x_0,y) = 0 \wedge \mathbf{b}_{i-1}(x_0,y) = 0\right] \geq 2/3 - \operatorname{neg}(\kappa)$. Hence,

$$\begin{aligned}&\Pr\left[\mathbf{b}_{i-1}(x_0,y) = 0\right] = \\ &\Pr_{y\leftarrow Y}\left[\mathbf{a}_i(x_0,y) = 1 \wedge \mathbf{b}_{i-1}(x_0,y) = 0\right] + \Pr\left[\mathbf{a}_i(x_0,y) = 0 \wedge \mathbf{b}_{i-1}(x_0,y) = 0\right] \\ &\geq 2/3 + 1/\operatorname{poly}(\kappa).\end{aligned}$$

and thus $C^{P_1}(x)$ violates correctness, which we have ruled out by assumption. □

Claim 5.4. *Suppose that* $\Pr\left[\mathbf{a}_i(x_2,y) \neq \mathbf{b}_{i-1}(x_2,y)\right] \geq 1/\operatorname{poly}(\kappa)$*, for some* $y \in Y$*. Then, either* $A^{P_2}(y)$ *violates privacy, or* $A^{P_1}(x)$ *violates privacy, for some* $x \in \{x_0, x_1\}$*, or* $C^{P_1}(x)$ *violates corectness, for some* $x \in \{x_0, x_1\}$*.*

Proof of Claim 5.4. We distinguish two cases depending on whether $y = y_2$.

Case 1. Without loss of generality, assume that $\Pr\left[\mathbf{a}_i(x_2,y_0) \neq \mathbf{b}_{i-1}(x_2,y_0)\right] \geq 1/\operatorname{poly}(\kappa)$. We proceed under the assumption that $C^{P_1}(x)$ does not violate correctness and $A^{P_1}(x)$ does not violate privacy, for $x \in \{x_0, x_1\}$. By Claim 5.3, $\Pr\left[\mathbf{a}_i(x,y_0) \neq \mathbf{b}_{i-1}(x,y_0)\right] \leq \operatorname{neg}(\kappa)$, for $x \in \{x_0, x_1\}$. By the hypothesis of the theorem and by correctness, the following is noticeable:

1. $\Pr\left[\mathbf{a}_i(x_2,y_0) = 0 \wedge \mathbf{b}_{i-1}(x_2,y_0) = 1\right]$

The above enables $A^{P_2}(y_0)$ to violate privacy in an execution where P_1 chooses his input uniformly at random (Since, by Claims 5.2 and 5.3, $\mathbf{a}_i = 0$ and $\mathbf{b}_{i-1} = 1$ implies that P_1 is holding input x_2).

Case 2. Assume that $\Pr_{x\leftarrow X}\left[\mathbf{a}_i(x,y_2) \neq \mathbf{b}_{i-1}(x,y_2)\right] \geq 1/\operatorname{poly}(\kappa)$. We proceed under the assumption that $C^{P_1}(x)$ does not violate correctness and $A^{P_1}(x)$ does not violate privacy, for $x \in \{x_0, x_1\}$. By Claim 5.3, $\Pr\left[\mathbf{a}_i(x,y_0) \neq \mathbf{b}_{i-1}(x,y_0)\right] \leq \operatorname{neg}(\kappa)$, for $x \in \{x_0, x_1\}$. Therefore, by the hypothesis of the theorem, the following is noticeable:

1. $\Pr\left[\mathbf{a}_i(x_2,y_2) = 2 \wedge \mathbf{b}_{i-1}(x_2,y_2) = 0\right] = \Pr\left[\mathbf{b}_{i-1}(x_2,y_2) = 0\right]$

Therefore,

$$\begin{aligned}\frac{2}{3} &= \Pr_{y \leftarrow Y}\left[\mathbf{b}_{i-1}(x_2, y) = 0\right] \\ &= \frac{1}{3} \cdot \left(\Pr\left[\mathbf{b}_{i-1}(x_2, y_0) = 0\right] + \Pr\left[\mathbf{b}_{i-1}(x_2, y_1) = 0\right] + \Pr\left[\mathbf{b}_{i-1}(x_2, y_2) = 0\right]\right)\end{aligned}$$

Thus, we deduce that $\Pr\left[\mathbf{b}_{i-1}(x_2, y') = 0\right] \leq 1 - 1/\operatorname{poly}(\kappa)$ for some $y' \in \{y_0, y_1\}$ and thus $\Pr\left[\mathbf{b}_{i-1}(x_2, y') \neq \mathbf{a}_i(x_2, y')\right] \geq 1/\operatorname{poly}(\kappa)$, which boils down to the previous case. □

□

Acknowledgments

We are grateful to Noam Mazor, Matan Orland and Jad Silbak for helpful discussions.

Y. Ishai and E. Kushilevitz were supported by ISF grant 1709/14, NSF-BSF grant 2015782, and a grant from the Ministry of Science and Technology, Israel and Department of Science and Technology, Government of India. Y. Ishai and N. Makriyannis were additionally supported by ERC Project NTSC (742754).

Bibliography

[1] Navneet Agarwal, Sanat Anand, and Manoj Prabhakaran. Uncovering algebraic structures in the MPC landscape. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, pages 381–406, 2019.

[2] Gilad Asharov. Towards characterizing complete fairness in secure two-party computation. In Yehuda Lindell, editor, *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014*, volume 8349 of *Lecture Notes in Computer Science*, pages 291–316. Springer, 2014.

[3] Gilad Asharov, Amos Beimel, Nikolaos Makriyannis, and Eran Omri. Complete characterization of fairness in secure two-party computation of boolean functions. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, 2015, Proceedings, Part I*, volume 9014 of *Lecture Notes in Computer Science*, pages 199–228. Springer, 2015.

[4] Donald Beaver. Multiparty protocols tolerating half faulty processors. In *Advances in Cryptology – CRYPTO '89*, pages 560–572, 1989.

[5] Donald Beaver and Shafi Goldwasser. Multiparty computation with faulty majority. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 468–473, 1989.

[6] Amos Beimel, Ariel Gabizon, Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, and Anat Paskin-Cherniavsky. Non-interactive secure multiparty computation. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, pages 387–404, 2014.

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

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

[9] David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols (extended abstract). In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC)*, pages 11–19, 1988.

[10] Benny Chor and Eyal Kushilevitz. A zero-one law for boolean privacy. *SIAM J. on Discrete Math.*, 4(1):36–47, 1991.

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

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

[13] Deepesh Data and Manoj Prabhakaran. Towards characterizing securely computable two-party randomized functions. In *Publick Key Cryptography (PKC)*, pages 675–697, 2018.

[14] Vanesa Daza and Nikolaos Makriyannis. Designing fully secure protocols for secure two-party computation of constant-domain functions. In *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*, pages 581–611, 2017.

[15] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 554–563, 1994.

[16] Matthias Fitzi, Juan A. Garay, Ueli M. Maurer, and Rafail Ostrovsky. Minimal complete primitives for secure multi-party computation. *J. Cryptology*, 18(1):37–61, 2005.

[17] Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. On 2-round secure multiparty computation. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, pages 178–193, 2002.

[18] Oded Goldreich. *Foundations of Cryptography — Basic Applications.* Cambridge University Press, 2004.

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

[20] Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In *Advances in Cryptology – CRYPTO '90*, pages 77–93, 1990.

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

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

[23] S. Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. Complete fairness in secure two-party computation. *J. ACM*, 58(6):24:1–24:37, 2011.

[24] Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. On combining privacy with guaranteed output delivery in secure multiparty computation. In *Advances in Cryptology – CRYPTO 2006*, pages 483–500, 2006.

[25] Yuval Ishai, Jonathan Katz, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. On achieving the "best of both worlds" in secure multiparty computation. *SIAM Journal on Computing*, 40 (1):122–141, 2011.

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

[27] Eyal Kushilevitz. Privacy and communication complexity. *SIAM J. on Discrete Math.*, 5(2): 273–284, 1992.

[28] Yehuda Lindell and Tal Rabin. Secure two-party computation with fairness - A necessary design principle. In *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*, pages 565–580, 2017.

[29] Nikolaos Makriyannis. On the classification of finite boolean functions up to fairness. In Michel Abdalla and Roberto De Prisco, editors, *Security and Cryptography for Networks - 9th International Conference, SCN 2014, 2014.*, volume 8642 of *Lecture Notes in Computer Science*, pages 135–154. Springer, 2014.

[30] Tal Rabin and Michael 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.

[31] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 162–167, 1986.

A Appendix

A.1 Feasibility landscape of Boolean solitary functionalities (Cont'd)

For every $M \in \{0,1\}^{n\times n}$, define $\mathcal{S}_M = \{\mathbf{y} = M\mathbf{x} \mid \sum_i \mathbf{x}_i \leq 0\}$.

Conjecture A.1. *It holds that* $\Pr_M[\mathbf{1}_n \in \mathcal{S}_M] \in o(1)$.

To support the above conjecture, for each value of $n \in \{4, \ldots, 300\}$, we sampled 10000 random binary matrices to check whether the condition is satisfied. The probability does appear to vanish, albeit quite slowly.

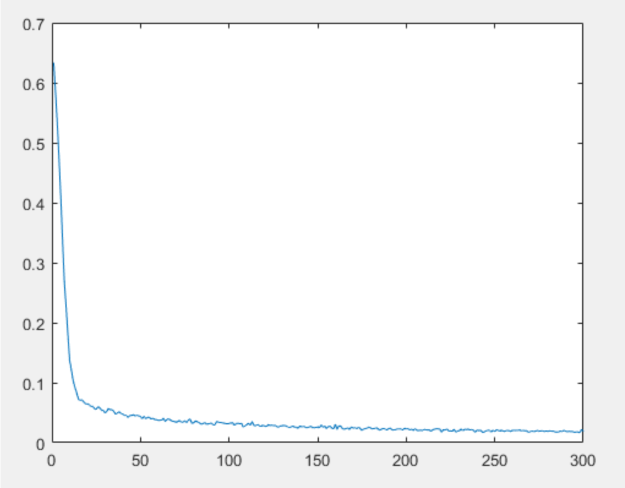

Figure 8: Experimental upper bound of $\Pr_M[\mathbf{1}_n \in \mathcal{S}_M]$.

A.2 Sketch argument for the necessity of Broadcast

Let $f : \{x_0, x_1\} \times \{y_1, y_2\} \to \{0,1,2\}$ such that $f(x_i, y_j) = i \cdot j$. Suppose that P_1 and P_2 provide input and Q is the output receiving party. The associated truth table has the following form.

$$\begin{pmatrix} 1 & 2 \\ 0 & 0 \end{pmatrix}$$

Suppose that f admits a fully secure protocol without broadcast. Now consider the following two attacks: (**Attack 1**) P_1 is corrupt and never sends any messages to P_2. At the same time, P_1 behaves seemingly honestly with Q as if P_2 never sent a message. (**Attack 2**) P_2 is corrupt and never sends any messages to P_1. At the same time, P_2 behaves seemingly honestly with Q as if P_1 never sent a message.

Because the protocol is assumed to be fully secure, at the end of the execution, party Q outputs a value that should be (1) consistent with P_2 input in the first case, and (2) consistent with P_1 input in the second. However, from Q's perspective, the executions above are indistinguishable which means that Q's output is consistent with both inputs (i.e. it's the correct output). In turn, this implies that f can be computed with full security even when all communication goes through Q.

However, in that case, a corrupt Q can arbitrarily change the input of P_1, which violates the privacy of P_2.

A.3 Tables With Examples

A.3.1 Equality Testing with Arbitrary leakage

Recall that $\mathsf{EqTst}^g_m(x, y) \mapsto 0$ if $x \neq y$ and $g(x)$ otherwise, for some Functionality g such that $0 \notin \text{im}(g)$. Recall that $\mathsf{EqTst}^g_m(x, y)$ admits trivial leakage if (1) $|\text{im}(g)| = 1$ or (2) $|\text{im}(g)| = 2$ and $|g^{-1}(\sigma)| = 1$, for some $\sigma \in \text{im}(g)$.

Input param.	Feasibility Criterion
Trivial leakage (1)	Forced
Trivial leakage (2)	Outlier Protocol
Nontrivial leakage	Impossible

A.3.2 PSI Variants

Same Fixed-sized Input PSI. $\mathsf{PSI}^g_{m,k}$ takes as inputs sets of size k from universe $[m]$ and applies Functionality g to the output.

Input param. \ Functionality g	$S \mapsto S$	$S \mapsto \lvert S \rvert$	$S \mapsto \begin{cases} 1 & \text{if } S \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$
$k < m/2$	Impossible	Forced	Forced
$k \geq m/2$	Open	Forced	Forced

Diff. Fixed-sized Input PSI. $\mathsf{PSI}^g_{m,k_1,k_2}$ takes as inputs sets of size k_1 and k_2 from universe $[m]$ and applies Functionality g to the output.

Input param. \ Functionality g	$S \mapsto S$	$S \mapsto \lvert S \rvert$	$S \mapsto \begin{cases} 1 & \text{if } S \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$
Input param. $k_1 \neq k_2$	Open	Forced	Forced

Same Range-Input PSI. $\mathsf{PSI}^g_{m,k_{\text{max}},k_{\text{min}}}$ takes as inputs sets of size from k_{max} and k_{min} from universe $[m]$ and applies Functionality g to the output.

Input param. \ Functionality g	$S \mapsto S$	$S \mapsto \lvert S \rvert$	$S \mapsto \begin{cases} 1 & \text{if } S \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$
$k_{\text{min}} = 0$	Forced	Forced	Forced & Fairness
$k_{\text{max}} = n$	Revealing	Open	Forced & Fairness
$k_{\text{max}} + k_{\text{min}} \leq n$	Open	Open	Open
$k_{\text{max}} + k_{\text{min}} > n$	Open	Open	Forced & Fairness

Diff. Range-Input PSI. $\mathsf{PSI}^g_{m,r^1,r^2}$ takes as inputs sets of size, respectively, from r^1_{min} and r^1_{max}, and r^1_{min} and r^2_{max} from universe $[m]$ and applies Functionality g to the output. Write $\mu_\ell = \sum_{i=r^\ell_{\text{min}}}^{r^\ell_{\text{max}}} \binom{m}{i}$ i.e. μ_ℓ is the number of inputs for P_ℓ.

Input param. \ Functionality g	$S \mapsto S$	$S \mapsto \lvert S \rvert$	$S \mapsto \begin{cases} 1 & \text{if } S \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$
$\mu_1 \neq \mu_2$	Open	Open	Fairness

A.3.3 OT Flavors

Standard OT. Let $\mathsf{OT}^g : (I, (\sigma_1, \ldots, \sigma_m)) \mapsto g(I, \sigma_I)$ takes as inputs sets of indices $I \subset [m]$ and a database of m elements from some space Σ and applies a functionality g to (I, σ_I).

Input param. \ Functionality g	Identity	$(I, \sigma_I) \mapsto \sigma_I$
$\lvert I\rvert = 1$	Revealing	Forced
$\lvert I\rvert = k$	Revealing	Outlier Protocol
Arbitrary I	Open	Open

Symmetric OT. Let $\mathsf{SymOT}^g : (I_1, (\sigma_1^1, \ldots, \sigma_m^1), I_2, (\sigma_1^2, \ldots, \sigma_m^2)) \mapsto g(I_1, \sigma_{I_1}^2, I_2, \sigma_{I_2}^1)$. Open for any interesting parameters.

A.3.4 Majority

Both of these functionalities are computable thanks to the Fairness criterion.

- 3-Party Majority where one of the parties receives output.
- 3-Party Majority where fourth party receives output.

All other cases are Open.

A.3.5 Return Max/Min

The n-party solitary functionality that returns the minimum (or maximum) of some input values is computable with full security thanks to the forced criterion.

A.3.6 Millionaires' Problem

$$f(x_1, \ldots, x_n) \mapsto i \text{ if } \forall j,\ x_i \geq x_{i-j} \wedge x_i > x_{i+j}$$

For $n = 2$ it follows from the forced output criterion. For larger n, it is unclear what the answer is.

n \ Party Q	$Q \in \{P_1, \ldots, P_n\}$	$Q \notin \{P_1, \ldots, P_n\}$
$n = 2$	Trivial	Forced/Fairness
$n > 3$	Open	Open

B Formal Description and Security Analysis of Outlier Protocol

Theorem B.1 (Restatement of Theorem 4.7)**.** *Protocol* Π *computes* f *with statistical full security in the* ShrGn_f^**-hybrid model with identifiable abort.*

Functionality ShrGn^*_f

- **Input:** P_1 holds $x \in X$ and 1^κ, P_2 holds $y \in Y$ and 1^κ, party Q holds 1^κ.

- **Computation:**

1. If Q alone does not provide input, or any two parties do not provide input, then the functionality returns a special default value def to the parties.

2. If P_1 alone does not provide input

   (a) An input $\widetilde{x} \leftarrow \{x_0, x_1\}$ is chosen uniformly at random.

   (b) Sample $b_0^{\#P_2} \leftarrow \{0,1,2\}$ and compute $b_0^{\#Q} = f(\widetilde{x}, y) - b_0^{\#P_2}$.

3. If P_2 alone does not provide input

   (a) An input $\widetilde{y} \leftarrow \{y_0, y_1\}$ is chosen uniformly at random.

   (b) Sample $a_0^{\#P_1} \leftarrow \{0,1,2\}$ and compute $a_0^{\#Q} = f(x, \widetilde{y}) - a_0^{\#P_1}$.

4. Otherwise

   (a) Compute $r = r(\kappa)$.

   (b) Sample an integer $i^* \geq 1$ chosen according to the geometric distribution with parameter $1/3$, i.e. the index of the first success in a sequence of Bernoulli trials with parameter $1/3$ corresponds to i^*. Set $i^* = \perp$ if the experiment does not succeed in the first r steps.

   (c) Compute two sequences of r values $(\widetilde{x}_1, \ldots, \widetilde{x}_r)$ and $(\widetilde{y}_1, \ldots, \widetilde{y}_r)$ such that $\widetilde{x}_i$ and $\widetilde{y}_i$ uniformly at random from $\{x_0, x_1\}$ and $\{y_0, y_1\}$, for $i < i^*$, and $\widetilde{x}_i = x$ and $\widetilde{y}_i = y$, for $i \geq i^*$.

   (d) Compute three sequences of $2r$ numbers $\{(a_1^{\#S}, b_1^{\#S}, \ldots, a_r^{\#S}, b_r^{\#S})\}_{S \in \{P_1, P_2, Q\}}$ such that

      i. $a_i^{\#P_1} \leftarrow \{0,1,2\}$ and $b_i^{\#P_1} \leftarrow \{0,1,2\}$ uniformly at random.

      ii. $a_i^{\#P_2} \leftarrow \{0,1,2\}$ and $b_i^{\#P_2} \leftarrow \{0,1,2\}$ uniformly at random.

      iii. $a_i^{\#Q} = f(x, \widetilde{y}_i) - a_i^{\#P_1} - a_i^{\#P_2}$ and $b_i^{\#Q} = f(\widetilde{x}_i, y) - b_i^{\#P_1} - b_i^{\#P_2}$.

- **Output:**

Case 1. The functionality returns a special default value def to the parties.

Case 2. Party P_2 receives $b_0^{\#P_2}$ and Q receives $b_0^{\#Q}$.

Case 3. Party P_1 receives $a_0^{\#P_1}$ and Q receives $a_0^{\#Q}$.

Case 4. Each $S \in \{P_1, P_2, Q\}$ receives $(a_1^{\#S}, b_1^{\#S}, \ldots, a_r^{\#S}, b_r^{\#S})$.

Figure 9: Three-Party Functionality ShrGn^*_f.

Protocol Π in the ShrGn_f^*-Hybrid Model with Identifiable Abort

- **Input:** P_1 holds $x \in X$ and 1^κ, P_2 holds $y \in Y$ and 1^κ, party Q holds 1^κ.

1. Q compute $c = f(\widetilde{x}, \widetilde{y})$ where $\widetilde{x}$ and $\widetilde{y}$ are chosen uniformly and independently at random from X and Y respectively.

2. The parties invoke ShrGn_f^* functionality on their respective inputs. We distinguish four cases depending on the output of the invocation.

   Case 1. If alive, Q outputs c.

   Case 2. P_2 sends $b_0^{\#P_2}$ to Q who outputs $b_0^{\#Q} + b_0^{\#P_2}$.

   Case 3. P_1 sends $a_0^{\#P_1}$ to Q who outputs $a_0^{\#Q} + a_0^{\#P_1}$.

   Case 4. Each $S \in \{P_1, P_2, Q\}$ receives $(a_1^{\#S}, b_1^{\#S}, \ldots, a_r^{\#S}, b_r^{\#S})$ and proceed to Step 3.

   Abort scenarios:

   - If Q aborts, then the computation halts.
   - If P_1 (resp. P_2) does not send its message in Case 3 (resp. Case 2) then Q output c.
   - If the computation was aborted in Case 4 then the remaining parties Q and P_1 (or Q and P_2) repeat the present step.

3. For $i = 1 \ldots r$, P_1 sends $b_i^{\#P_1}$ to Q, and P_2 sends $a_i^{\#P_2}$ to Q.

   Abort scenarios:

   - If Q aborts, the computation halts.
   - If $i = 1$ and P_ℓ alone does not send its message, then parties $P_{3-\ell}$ and Q repeat Step 2.
   - If $i \neq 1$ and P_1 (resp. P_2) alone does not send its message, then P_2 (resp. P_1) sends $b_{i-1}^{\#P_2}$ (resp. $a_{i-1}^{\#P_1}$) to Q. Party Q outputs $b_{i-1}^{\#Q} + b_{i-1}^{\#P_1} + b_{i-1}^{\#P_2}$ (resp. $a_{i-1}^{\#Q} + a_{i-1}^{\#P_1} + a_{i-1}^{\#P_2}$). If the latter fails, refer to the next case.
   - If both P_1 and P_2 abort the computation, then Q outputs c.

- **Output:** Q's output is determined by the above. Parties P_1 and P_2 output nothing.

Figure 10: Protocol Π for computing f in the ShrGn_f^*-hybrid model

Proof. As discussed above, we only address the case where the adversary both P_1 and Q and sends x_0 for the computation (x_1 is analogous and the remaining case, i.e. x_2, is easy). Further, similarly to the simulation of the GHKL protocol [23], it suffices to prove the theorem for an adversary aborting at round $i \leq i^*$ (since the complement case $i > i^*$ yields identical distributions for the real and ideal world). The simulator is fully described in Figure 11 on p. 35. Finally, it suffices to analyze the distribution of the pair (a_i, b_{i-1}), where i is the aborting round. The distribution of the pair is described by the table below. We write α instead of $1/3$ to make the table more legible. We write the probabilities as vectors to account for every input of an honest P_2. A straightforward calculation yields that both worlds are identically distributed.

$\Pr[(a_i, b_{i-1})]$	Real	Ideal
$(0,0)$	$\alpha \cdot (0, 1/2, 1) + \frac{1-\alpha}{2} \cdot (1/2, 1/2, 1)$	$\alpha \cdot (0, 1, 1) + \frac{1-\alpha}{2} \cdot (\frac{(1,0,1)}{2} + \frac{(0,0,1)}{2})$
$(0,1)$	$\alpha \cdot (0, 1/2, 0) + \frac{1-\alpha}{2} \cdot (1/2, 1/2, 0)$	$\alpha \cdot (0, 0, 0) + \frac{1-\alpha}{2} \cdot (\frac{(0,1,0)}{2} + \frac{(1,1,0)}{2})$
$(1,0)$	$\alpha \cdot (1/2, 0, 0) + \frac{1-\alpha}{2} \cdot (1/2, 1/2, 1)$	$\alpha \cdot (1/2, 0, 0) + \frac{1-\alpha}{2} \cdot (1/2, 1/2, 1)$
$(1,1)$	$\alpha \cdot (1/2, 0, 0) + \frac{1-\alpha}{2} \cdot (1/2, 1/2, 0)$	$\alpha \cdot (1/2, 0, 0) + \frac{1-\alpha}{2} \cdot (1/2, 1/2, 0)$

□

Simulator S

Let x denote the input chosen by the adversary.

- **Case** $x = x_0$**.** Choose i^* according to the prescribed distribution.
  - **For** $i = 0 \ldots i^* - 1$ set $a_i \leftarrow \{0,1\}$ and reveal it to the adversary. If the adversary aborts, do as follows.
    - ∗ If $a_i = 1$ then invoke the trusted party on input x_2 and set

      $$b_{i-1} = \begin{cases} \widetilde{b}_{i-1} \leftarrow \{0,1\} & \text{if } f(x_2, y) = 0 \\ 0 & \text{othwerwise.} \end{cases}$$

      output $(a_0, \ldots, a_i, b_{i-1})$ and halt.
    - ∗ If $a_i = 0$ then invoke the trusted party on input $\hat{x} \leftarrow \{x_1, x_2\}$ and set

      $$b_{i-1} = \begin{cases} f(\hat{x}, y) & \text{if } \hat{x} = x_1 \\ 1 & \text{if } \hat{x} = x_2 \text{ and } f(x_2, y) = 0 \\ 0 & \text{if } \hat{x} = x_2 \text{ and } f(x_2, y) = 2 \end{cases}$$

      output $(a_0, \ldots, a_i, b_{i-1})$ and halt.
  - At iteration i^*, invoke the trusted party on input x_0 and set $a_{i^*} = f(x_0, y)$ and reveal it to the adversary. If the adversary aborts, set $b_{i^*-1} = 0$ if $a_{i^*} = 0$ and $b_{i^*-1} \leftarrow \{0,1\}$ otherwise, output $(a_0, \ldots, a_i, b_{i-1})$ and halt.
  - **For** $i = i^* + 1 \ldots r$ set $a_i = b_{i-1} = f(x_0, y)$. If the adversary aborts, output $(a_0, \ldots, a_i, b_{i-1})$ and halt.
- **Case** $x = x_1$**.** Analogous to the above modulo a permutation of the inputs (swap x_0 and x_1).
- **Case** $x = x_2$**.** Invoke the trusted party to obtain $f(x, y)$. Choose i^* according to the prescribed distribution. Set

  $$a_i = \begin{cases} 0 & \text{if } i < i^* \\ f(x, y) & \text{othwerwise.} \end{cases}$$

  $$b_i = \begin{cases} \widetilde{b}_i \leftarrow \{0,1\} & \text{if } i < i^* \wedge f(x, y) \neq 2 \\ 0 & \text{if } i < i^* \wedge f(x, y) = 2 \\ f(x, y) & \text{othwerwise.} \end{cases}$$

  Reveal the a's one by one to the adversary. If the adversary aborts after seeing a_i, output $(a_0, \ldots, a_i, b_{i-1})$ and halt.

Figure 11: Simulation for a Fail-Stop adversary corrupting P_1 and Q